WinLicense Help

USER MANUAL

WinLicense

1 WinLicense

WinLicense is a powerful software protection system and licensing manager designed for software developers who wish to protect their applications against advanced reverse engineering and software cracking. WinLicense uses the SecureEngine® protection system to achieve its goals, making it really difficult to break using the traditional and newest cracking tools.

WinLicense has been designed to completely stop novice and advanced crackers from cracking an application. This will avoid a considerable loss of revenue from the distribution of cracked applications. Developers do not need any source code changes or programming experience to protect their applications with WinLicense.

WinLicense also offers a wide rage of powerful and flexible techniques that allow developers to securely distribute trial versions of their applications.

1.1 Why use WinLicense?

WinLicense has been designed with the newest and most powerful technology in software protections and trial/licensing control, SecureEngine®.

From the attacker point of view, WinLicense is completely different to traditional software protectors, due to its complex protection engine and its high priority code that allows supervising the whole system against possible attackers. From the software developer's point of view, WinLicense is quite easy to use and easily adapts its protection techniques and trial/licensing control to suit a developer's needs.

The following sections give a detailed explanation about the benefits of using WinLicense to protect/license your software.

- Scenarios for using WinLicense 17
- Comparing WinLicense with other protectors/licensing systems 18

1.1.1 Scenarios for using WinLicense

WinLicense uses the SecureEngine® protection system to cover a wide range of scenarios. SecureEngine® is the ideal solution in the following situations:

- **Protecting an application against modifications and software piracy:** SecureEngine® protects the integrity of an application by encrypting and decrypting its code at runtime, using revolutionary techniques that defeats any of the traditional or newest cracking tools.

- **Protecting an application against reverse engineering:** SecureEngine® uses a wide range of techniques to prevent reverse engineering. An attacker will not be able to use cracking tools to analyze the code of a protected application.

- **Protecting an application against unauthorized access:** SecureEngine® allows a developer to include password protection in an application. The application can only be executed with a valid User-Password. The SecureEngine® SDK offers external functions that allow developers to handle the management of new users and passwords.

- **Protecting an application against monitoring tools:** SecureEngine® includes the most advanced techniques to detect registry and file monitoring tools. Developers choose the desired option to finish the execution of their applications upon the detection of monitoring tools.

- **Making trial versions for an application:** WinLicense offers powerful and flexible techniques that allow developers to securely distribute trial version of their applications, allowing developers to interact with the trial status of their applications via an extended API.

- **Licensing an application:** WinLicense offers a flexible and powerful licensing system that allow developers to create a wide range of registration system into their applications. WinLicense provides an extended API that offers total freedom to manage license keys.

1.1.2 Comparing WinLicense with other protectors/licensing systems

Other software protectors and licensing system have important vulnerabilities, which prevent them from being a perfect solution to protect an application against reverse engineering or cracking. The following section identifies some of those vulnerabilities and shows how WinLicense resolves them.

Obsolete protection techniques

Most modern software protection systems use already broken techniques that are quite easy to bypass. Normally, an attacker will reuse the same proven tools that have been used over years to break protection systems. Often the attacker will release a global technique to attack every application protected by a specific protection system.

SecureEngine® uses new technology in software protection to ensure each protected application is unique thus preventing any cracking tool from being used to create a universal crack to your application.

Attackers are one step ahead of the protection system

When a software protection system has been broken, their authors implement patches to avoid a specific attack from being used again on new versions. Typically attackers will inspect the new changes that have been applied in the new version and will easily bypass them again. In this common scenario, attackers are always one step ahead from the protection system because the new applied patches can easily be identified and defeated.

SecureEngine® has a different approach to avoid this. If vulnerability is found the vulnerable object is quickly changed (due to the mutable technology used in SecureEngine) instead of releasing a patch against the specific threat. The new object, joined with the rest of the SecureEngine® objects, creates a completely new protection system. The benefits of this, when compared to common software protectors, is that attackers will have to reexamine the whole protection code to bypass the new changes.

Static trial and licensing control

Most of the other trial systems do not include a wide range of APIs to interact with the licensing system or to check and extend the current status of the trial period in a protected application. This makes developers narrow their creativity to create a flexible trial/licensing system for their customers and forces them to use a rigid registration system for their application.

WinLicense comes with an extensive SDK to satisfy developers' needs, giving developers the freedom to hande all those situations that they want and leaving to WinLicense the hard work of those situations not handled by the developer.

Trial periods are easy to reset by attackers

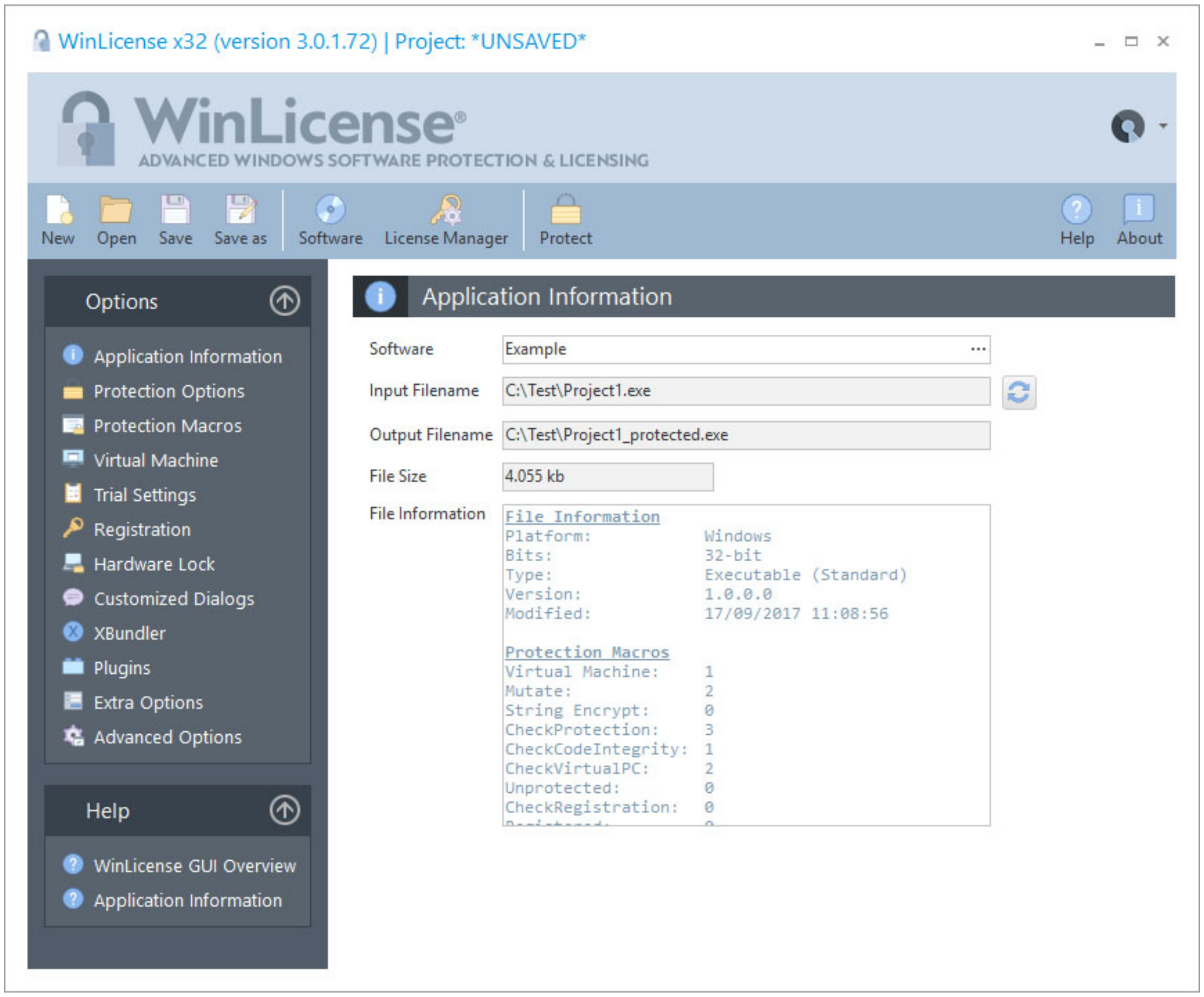
Most of the current licensing systems store the current trial status (days left, executions left, etc.) in places which are easy to find by attackers. Normally, the trial information is stored in the Windows registry and/or files. In this scenario, an attacker will use specific tools to compare the registry/files before and after executing a protected application. This shows them where the trial information is stored in the system and therefore, how to reset the trial period for an application.

WinLicense comes with the Trial Storing Manager which is a specialized technology to store the trial information of a protected application in random and special locations in the system. This makes the trial period of protected applications very difficult to reset.

1.2 User Interface

1.2.1 WinLicense GUI overview

WinLicense provides a flexible and easy to use user interface (GUI) to help you to protect your applications. WinLicense uses the classical project file approach to help you save and restore your settings each time you want to protect an application. We recommend that you first try to protect an application with default options (just create a new Software, select it and protect it). After that, you can add some Trial Restrictions and protect it, so you can know how the Trial restrictions work. Later you add Registration support and create a license for your protected application (from the License Manager 363) and see how your application gets registered.



Application Information

Software: Example

Input Filename: C:\Test\Project1.exe

Output Filename: C:\Test\Project1_protected.exe

File Size: 4.055 kb

File Information:

```
File Information
Platform:           Windows
Bits:               32-bit
Type:               Executable (Standard)
Version:            1.0.0.0
Modified:           17/09/2017 11:08:56

Protection Macros
Virtual Machine:    1
Mutate:             2
String Encrypt:     0
CheckProtection:    3
CheckCodeIntegrity: 1
CheckVirtualPC:     2
Unprotected:        0
CheckRegistration:  0
```

The Toolbars Menu

The toolbars menu helps you to manage your projects, softwares, customers, licenses and protect your application.

Every time that you make changes to your protection settings, you can save those changes into a project. That project can be loaded at a later time to restore the current protection settings and protect another software.

The Options Panel

The option panel shows the different settings that are required to protect an application and customize the different protection options that you want to include in your application.

- The Application Information 22 panel allows you to set up the general information settings about the application that you are going to protect.
- The Protection Options 24 panel allows you to select the different protection options that will be included in your application.
- The Protection Macros 27 panel allows you to review and/or select the specific protection macros that have been included in your application.
- The Virtual Machine 30 panel allows you to set up all protections related with the Virtual Machine technology.
- The Trial Settings 32 panel allows you to select the different trial settings that will be included in your application.
- The Registration 36 panel allows you to select the different registration settings that will be able to register your application.
- The Hardware Lock 40 option allows you to specify which hardware settings will be included in your hardware dependent keys.
- The Customized Dialogs 43 option allows you to customize the different messages that are shown when a certain event occurs while your protected application is running.
- The XBundler 48 panel allows you to embed data files and DLLs to be embedded inside your protected application.
- The Plugins 52 panel allows you to insert custom plugins (DLLs) that will be embedded inside the protected binary.

- The Extra Options 65 panel contains specific options (not protection) related that can be added into your application.

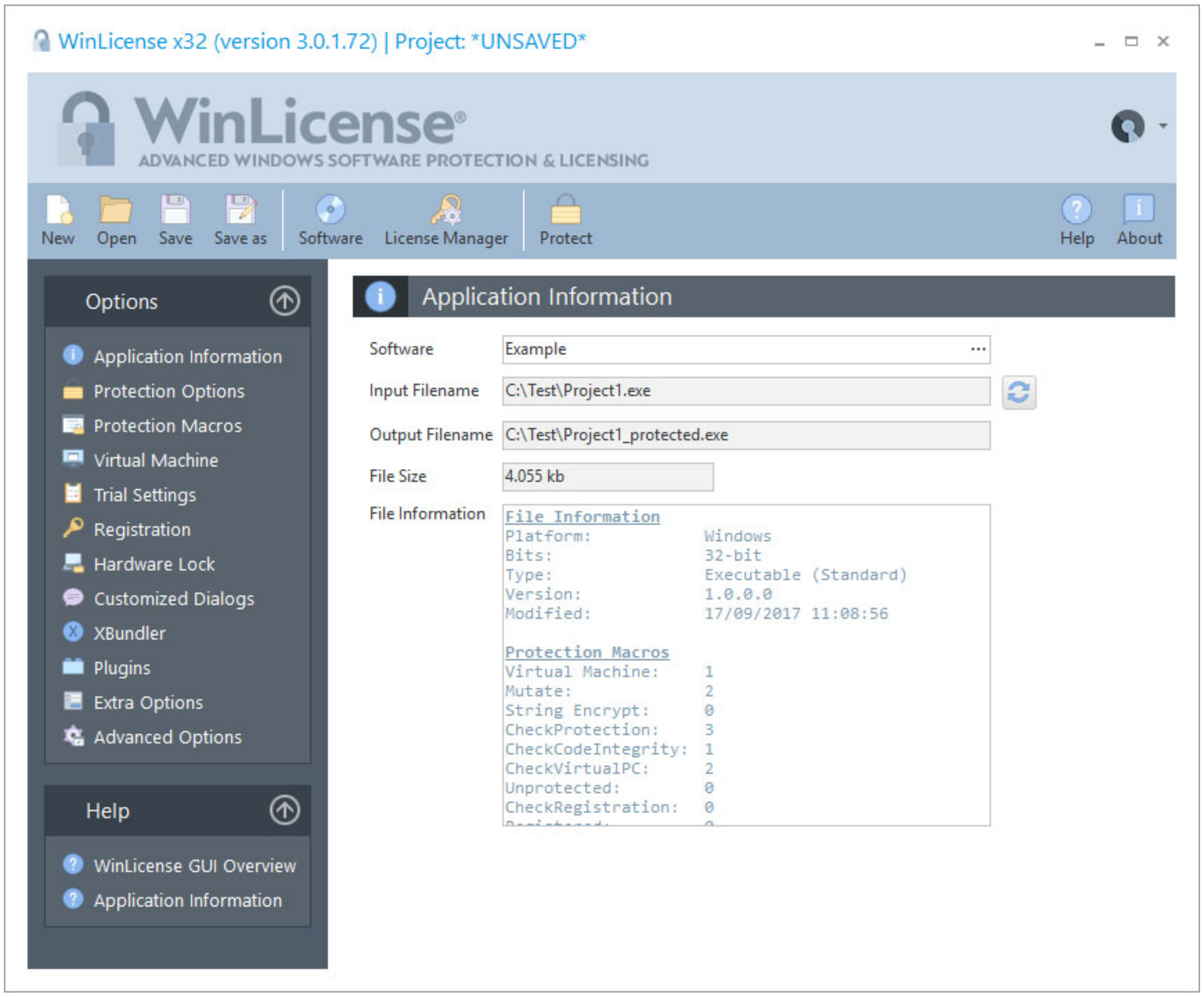
- The Advanced Options 67 panel allows you to insert special (hidden) options related with compatibility in specific applications.

The Help Panel

The help panel gives you access to the WinLicense Help file to get more information about specific topics.

1.3 Protecting an application

1.3.1 Application Information



Application Information

Software: Example

Input Filename: C:\Test\Project1.exe

Output Filename: C:\Test\Project1_protected.exe

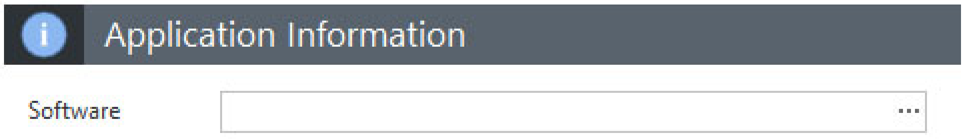
File Size: 4.055 kb

File Information

```
File Information
Platform:           Windows
Bits:               32-bit
Type:               Executable (Standard)
Version:            1.0.0.0
Modified:           17/09/2017 11:08:56

Protection Macros
Virtual Machine:    1
Mutate:             2
String Encrypt:     0
CheckProtection:    3
CheckCodeIntegrity: 1
CheckVirtualPC:     2
Unprotected:        0
CheckRegistration:  0
```

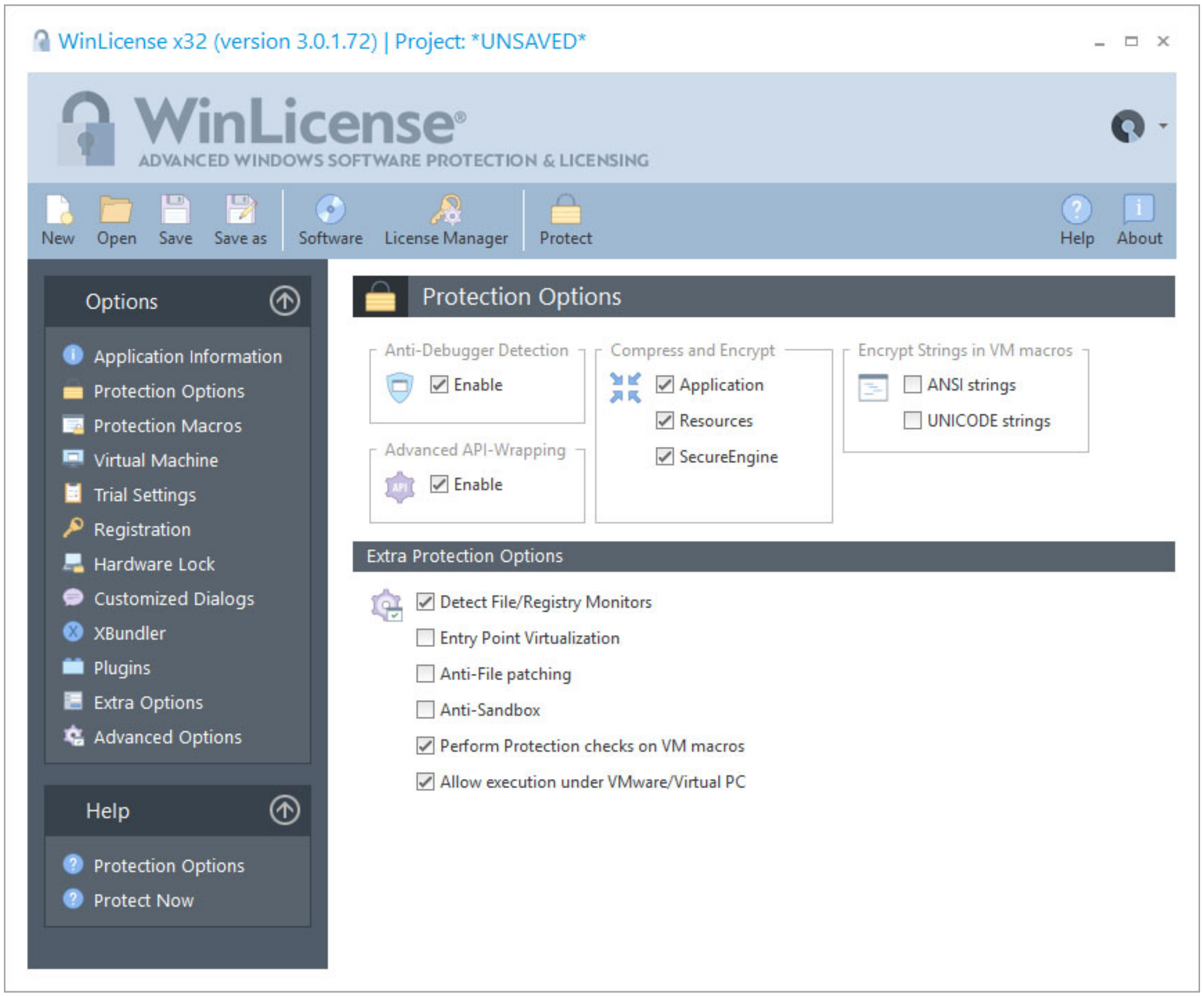
The first step to protecting an application is selecting the application that you want to protect by choosing a software from the Software Database. After selecting the software that you want to protect, related information about your software will be displayed in the Application Information panel.

From withing the Application Information panel, the first thing that you have to do is select a specific Software (which contains the input and output file names) to protect. You can click on the right button in the Software edit box.



Please, refer to the Software Panel 363 help topic for more information.

1.3.2 Protection Options



WinLicense x32 (version 3.0.1.72) | Project: *UNSAVED*

WinLicense®
ADVANCED WINDOWS SOFTWARE PROTECTION & LICENSING

Protection Options

Anti-Debugger Detection
- [✓] Enable

Advanced API-Wrapping
- [✓] Enable

Compress and Encrypt
- [✓] Application
- [✓] Resources
- [✓] SecureEngine

Encrypt Strings in VM macros
- [ ] ANSI strings
- [ ] UNICODE strings

Extra Protection Options
- [✓] Detect File/Registry Monitors
- [ ] Entry Point Virtualization
- [ ] Anti-File patching
- [ ] Anti-Sandbox
- [✓] Perform Protection checks on VM macros
- [✓] Allow execution under VMware/Virtual PC

In the Protection Options panel you can select the different protection options that you want to include in your application. By default all the protection options are enabled. If a specific protection option is not needed for your application it can be removed to speed up the execution of your application and make the protection code smaller.

Anti-Debugger Detection

This option will enable anti-debugger detections inside the protected application, detecting when a kernel or software debugger is debugging a protected application.

Advance API-Wrapping

This option will enable advanced API-Wrapping techniques that keep an attacker from identifying the different APIs that are used by a protected application. The API-Wrapping option has a very small penalty in the execution speed in your application in case that your application is calling a specific API massively. In any case, there are internal options that can help you to unselect specific functions from being wrapped. Please, contact us for further information.

Compress and Encrypt

You can select if your application, resources and the protection boot loader will be encrypted and compressed. There is a small penalty in the execution time before your application starts but it's recommended to keep those options enabled for further protection.

Encrypt Strings in VM macros

When you insert Virtual Machine macros in your source code (or via an external MAP file) you can encrypt all references to strings that appear inside the macro markers (START - END). The string will be removed from the original location and it will be moved inside the protection code area in an encrypted form. Once that the string is going to be referenced by your code, it will be decrypted at that specific point in order to deliver it to the required code.

If you are just using ANSI strings in your application, you should just check the "ANSI strings" options. If instead, your application is using UNICODE strings, you should just check the "UNICODE strings" option. You can go to the "Protection Macros 27" panel and select a specific macro from the list, after that click on the lower panel tabs (Ansi Strings and Unicode Strings) in order to see that strings that are found inside the selected protection macro.

This option is basically the same as putting a STR_ENCRYPT macro 80 inside each Virtual Machine Macro that you have inserted. If you are just interested in protecting specific strings that appears in specific macros that you have inserted, you should not use this option and use instead a STR_ENCRYPT macro 80 inside your Virtual Machine Macro.

Extra Protection Options

- **Detect File/Registry Monitors**: This option detects common tools that monitor access to the Windows File and Registry system by a specific application. If your application stores sensitive information in the Windows File or Registry system, you should enable this option.

- **Entry Point Obfuscation:** This option produces the equivalent result as putting a VM macro in the very first instructions that are executed in your application. This option is not compatible with all applications and you should uncheck it in case that your application does not start when protected.

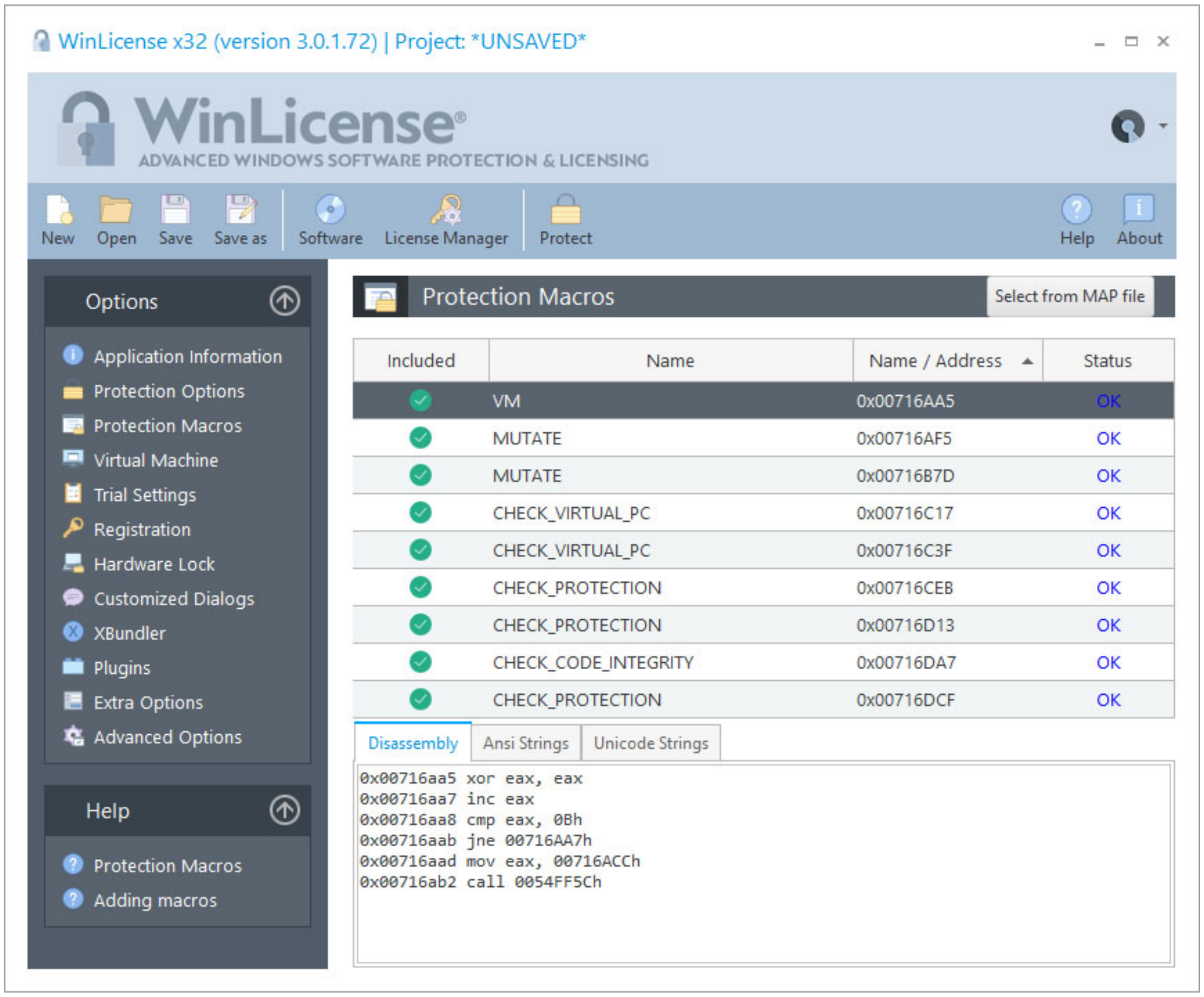
- **Anti-File Patching**: This options detects when a modification has been done to the protected application from an external source (like a virus, cracker or any other application). If you are planning to put another compressor on top of your protected application or do some external modifications to the protected file on disk, you should not check this option. If this option is set and and there is a modification in your protected application, SecureEngine will display the Customized Dialog 43 "MSG_ID_FILE_CORRUPTED". You can edit the error message or handle that specific error event from your plugin DLL.

- **Anti-Sandbox**: This option detects common sandbox applications. A sandbox application virtualizes file and registry access in order to avoid direct/real access to the file/registry system.

- **Perform Protection checks on VM macros**: When you insert a VM macro (TIGER VM, FISH VM, etc.) inside your application you can perform extra protection checks before the your protection macro is executed. This option checks if your application has been partially attacked by an attacker.

- **Allow execution under VMWare/Virtual PC**: This option allows your application to be run under common virtual environments like VMWare, Virtual PC, VirtualBox, etc. In case that you want to restrict the execution of your protected application under these virtual environments, you should uncheck this option.

1.3.3 Protection Macros



Included	Name	Name / Address	Status
✓	VM	0x00716AA5	OK
✓	MUTATE	0x00716AF5	OK
✓	MUTATE	0x00716B7D	OK
✓	CHECK_VIRTUAL_PC	0x00716C17	OK
✓	CHECK_VIRTUAL_PC	0x00716C3F	OK
✓	CHECK_PROTECTION	0x00716CEB	OK
✓	CHECK_PROTECTION	0x00716D13	OK
✓	CHECK_CODE_INTEGRITY	0x00716DA7	OK
✓	CHECK_PROTECTION	0x00716DCF	OK

```
0x00716aa5 xor eax, eax
0x00716aa7 inc eax
0x00716aa8 cmp eax, 0Bh
0x00716aab jne 00716AA7h
0x00716aad mov eax, 00716ACCh
0x00716ab2 call 0054FF5Ch
```

In the **Protection Macros** panel, you can see the assembly code that will be protected for each protection macro. You can also see if any ANSI or UNICODE strings are referenced inside each protection macro and decide if you want to protect those strings (see option **Encrypt Strings in VM macros** in the Protection Options 24 panel)

You can also enable or disable specific blocks from being protected. Normally, disabling macros from being protected is only required to find a problematic block in the protected application, which makes the application behave in a different way or produce an application exception. In case that you have problems protecting a specific macro, you should check that there is not a current macro restriction.

Macro Restrictions

Switch-case statements and *try-except* clauses cannot work with SecureEngine macros in most compilers.

Compilers generate a direct jump table in the data section which directly jumps to each "case" statement. When the code is virtualized, the jump goes into a virtualized (garbage) code and it produces exception. Switch-case and try-except clauses will be supported in a future version.

You can use a workaround to protect your switch-case statements with VM macros, like:

For switch-case:

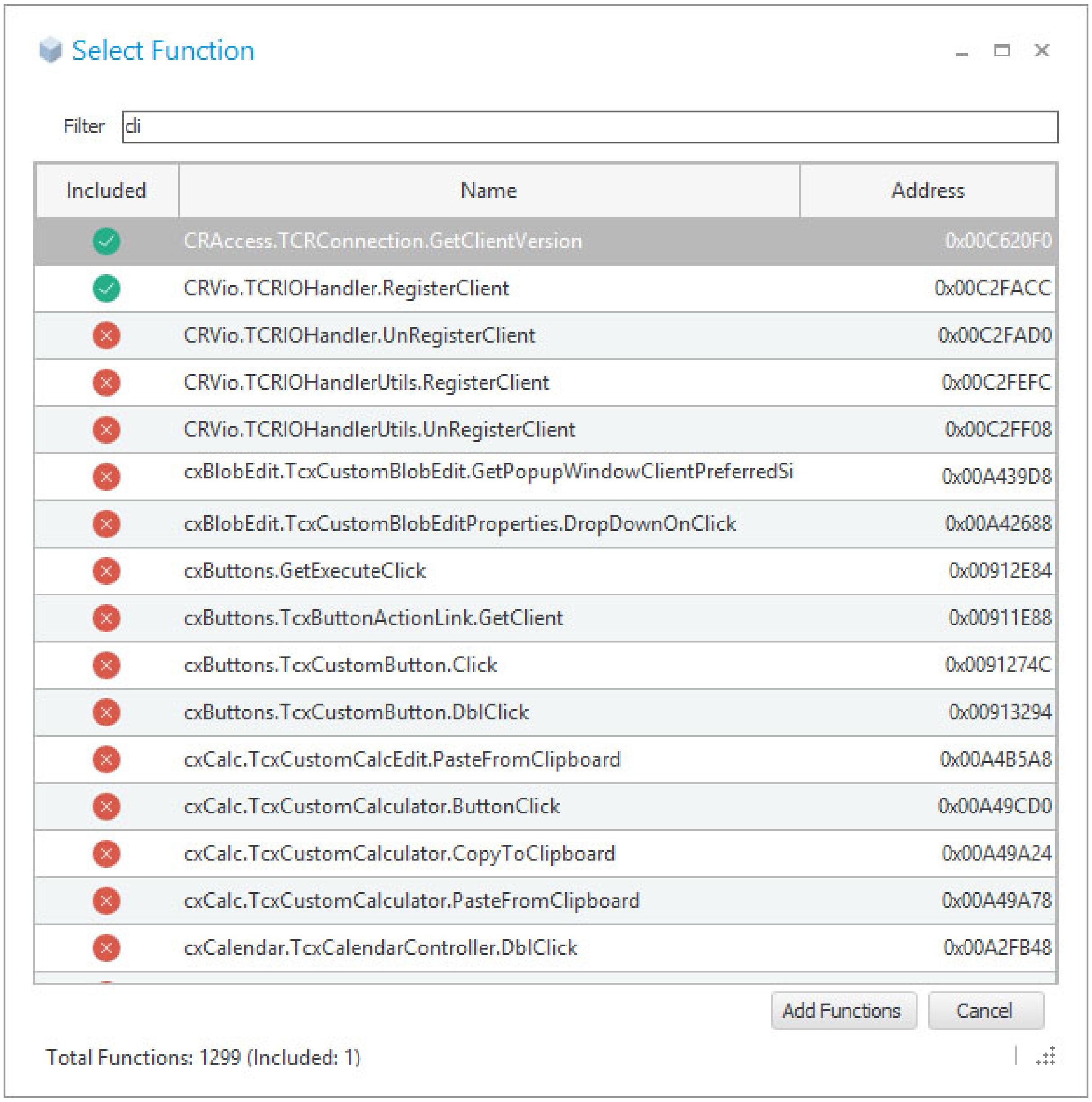
```
switch (var)
{
    case 0:

    VM_START

    // your code

    VM_END

    case 1:

    VM_START

    // your code

    VM_END

    ...
}
```

For try-except:

```
try
{
    VM_START

    // your code

    VM_END
}

except
{
    VM_START

    // your code

    VM_END
}
```

Inserting Macros from a MAP file

To insert macros from a MAP file, just click on the **Select from MAP file** button. You can insert/remove macros by clicking on the name of the function.



Filter: cli

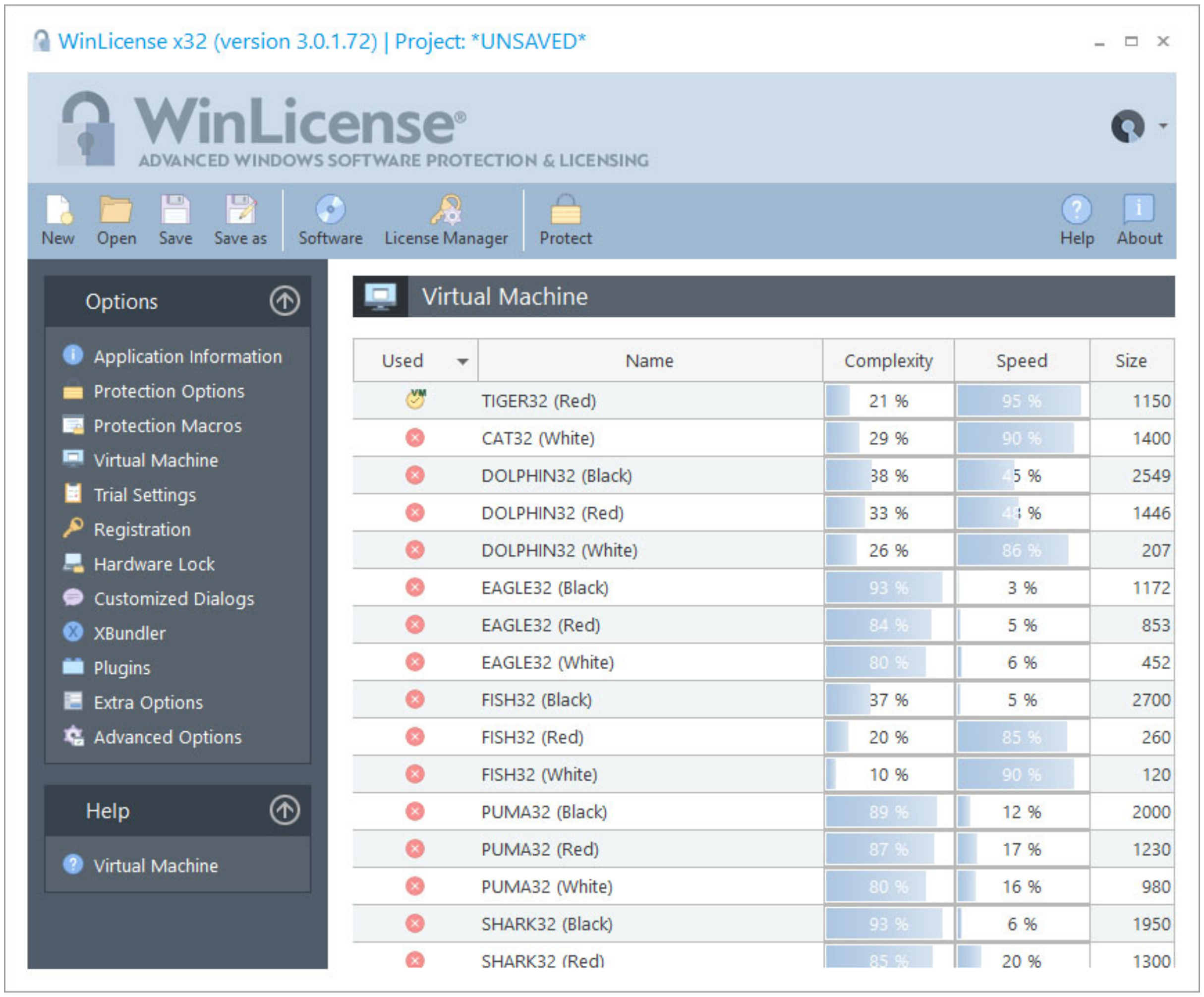
Included	Name	Address
✓	CRAccess.TCRConnection.GetClientVersion	0x00C620F0
✓	CRVio.TCRIOHandler.RegisterClient	0x00C2FACC
✗	CRVio.TCRIOHandler.UnRegisterClient	0x00C2FAD0
✗	CRVio.TCRIOHandlerUtils.RegisterClient	0x00C2FEFC
✗	CRVio.TCRIOHandlerUtils.UnRegisterClient	0x00C2FF08
✗	cxBlobEdit.TcxCustomBlobEdit.GetPopupWindowClientPreferredSi	0x00A439D8
✗	cxBlobEdit.TcxCustomBlobEditProperties.DropDownOnClick	0x00A42688
✗	cxButtons.GetExecuteClick	0x00912E84
✗	cxButtons.TcxButtonActionLink.GetClient	0x00911E88
✗	cxButtons.TcxCustomButton.Click	0x0091274C
✗	cxButtons.TcxCustomButton.DblClick	0x00913294
✗	cxCalc.TcxCustomCalcEdit.PasteFromClipboard	0x00A4B5A8
✗	cxCalc.TcxCustomCalculator.ButtonClick	0x00A49CD0
✗	cxCalc.TcxCustomCalculator.CopyToClipboard	0x00A49A24
✗	cxCalc.TcxCustomCalculator.PasteFromClipboard	0x00A49A78
✗	cxCalendar.TcxCalendarController.DblClick	0x00A2FB48

Total Functions: 1299 (Included: 1)

By default, the added MAP functions are handled by a "VM" macro. If you want to assign a specific virtual machine for each added MAP function, do the following steps:

- In the Protection Macros panel, select the specific MAP function
- Hold the **[SHIFT]** key and press the **[LEFT_ARROW]** or **[RIGHT_ARROW]** key to move between different machines

1.3.4 Virtual Machine



Used	Name	Complexity	Speed	Size
✓	TIGER32 (Red)	21 %	95 %	1150
✗	CAT32 (White)	29 %	90 %	1400
✗	DOLPHIN32 (Black)	38 %	45 %	2549
✗	DOLPHIN32 (Red)	33 %	48 %	1446
✗	DOLPHIN32 (White)	26 %	86 %	207
✗	EAGLE32 (Black)	93 %	3 %	1172
✗	EAGLE32 (Red)	84 %	5 %	853
✗	EAGLE32 (White)	80 %	6 %	452
✗	FISH32 (Black)	37 %	5 %	2700
✗	FISH32 (Red)	20 %	85 %	260
✗	FISH32 (White)	10 %	90 %	120
✗	PUMA32 (Black)	89 %	12 %	2000
✗	PUMA32 (Red)	87 %	17 %	1230
✗	PUMA32 (White)	80 %	16 %	980
✗	SHARK32 (Black)	93 %	6 %	1950
✗	SHARK32 (Red)	85 %	20 %	1300

The Virtual Machine panel allows you to integrate the Virtual Machine technology into your application.

Available Virtual Machines

This panel shows all the available Virtual Machines that can be used in a protected application. Some Virtual Machines are public and others are private for specific customers. You can contact us at info@oreans.com to know more about customized Virtual Machines.

Suppose that you protect the same application two times using the TIGER Virtual Machine. Each protected instance will contain a unique TIGER Virtual Machine with different registers, instruction handlers, opcode table, etc. from the previous instance. They will just share the internal skeleton of the TIGER architecture. A cracker will have to study the internal skeleton of the TIGER architecture an later try to find a way to to attack all different variations of the TIGER architecture. This scheme is the one that contains all current software protectors based on Virtual Machines (they use mutations/variations of an internal architecture model defined by them).

We wanted to go one step further, creating multiple Virtual Machine architectures with the help of our powerful Virtual Machines Generator tool. Comparing two different architecture names, like TIGER and LION, is equivalent to comparing an Intel x86 processor with an ARM processor. Each one is totally independent from each other and developed without the other in mind.

The **Complexity** and **Speed** columns display some stats about the execution speed and the complexity of a given Virtual Machine. Notice that depending on the internal Virtual Machine revision, those values might change (increasing or decreasing across versions).

Virtual Machine selected for the Protection Boot loader

The protection boot loader (the code executed before your application takes control) uses the internal virtualization engine to protect itself from being inspected. You can select a specific Virtual Machine that will virtualize the protection boot code. To do so, just right click on the specific Virtual Machine and select "Use it in Protection Boot". We recommend you not using a very complex virtual machine (with low speed) to avoid a noticeable performance decrease while your application is being loaded.

Virtual Machine selected for standard (old) VM macros

If you have inserted the old VM_START/END macro in your source code, you can associate a specific Virtual Machine name to those macros. To do so, just right-click on the specific machine and select "Use it for old VM macros"

Guidelines to select Virtual Machines

Developers might feel confused about which Virtual Machine they should select in order to get the desired security for their applications. Some developers might have the idea of adding as many Virtual Machines and CPUs as possible to highly increase the security of their applications. This might not produce the effect that they want.

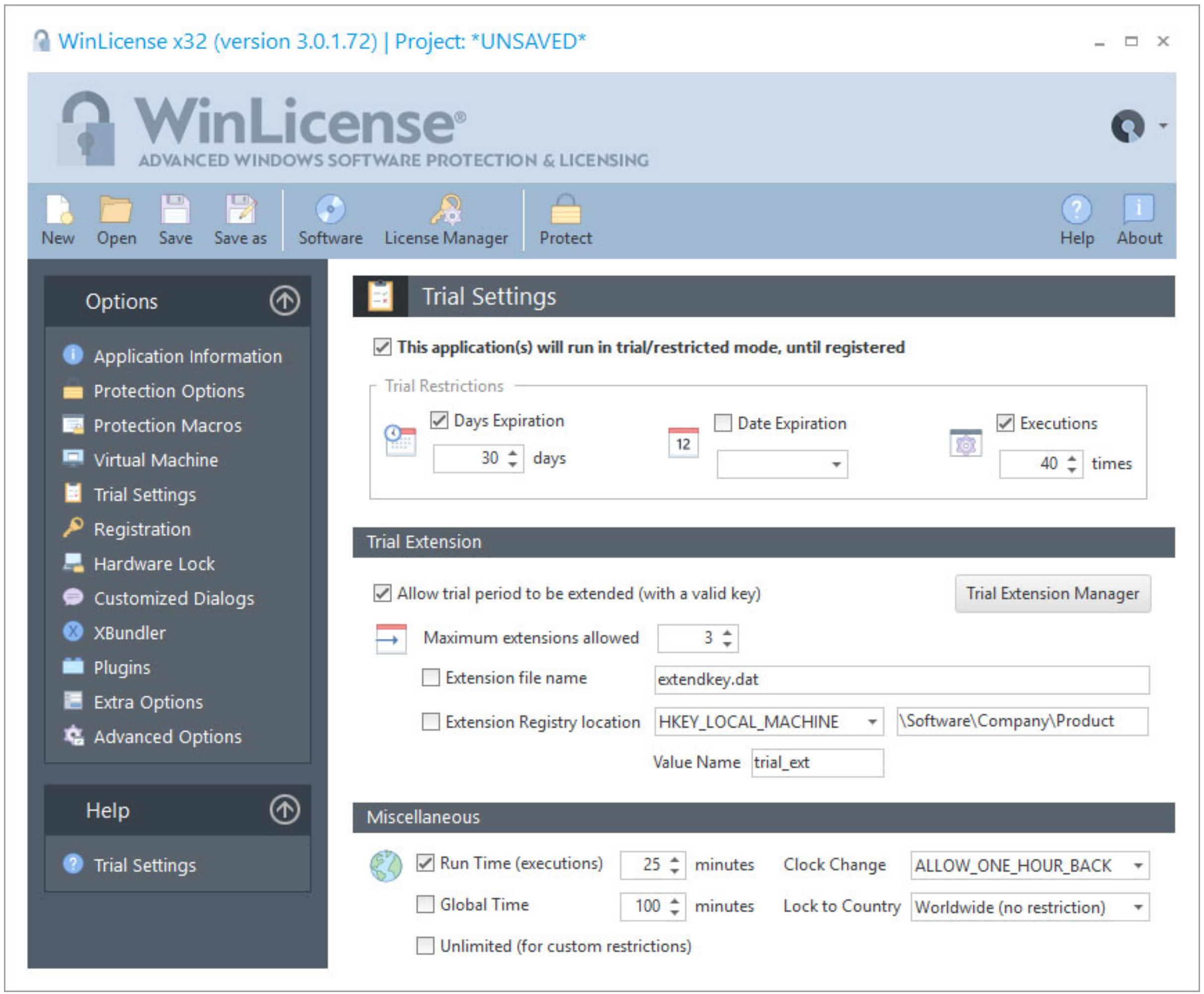
1) Just insert a single Virtual Machine or a couple of them. If you insert several Virtual Machines and CPUs, it will produce a big protected application on disk and memory, as some Virtual Machines can be bigger than 1Mb. Notice that if you select several Virtual Machines

and they are not selected in the lower "Virtualization" panel, SecureEngine will not insert those Virtual Machines inside your application, as they will be unused by the protection (optimizing the final size of the protected application)

2) When using the virtualization macros in your application, you should avoid using the old VM macro and specify the Virtual Machine architecture that will protect a specific macro (example: "VM_TIGER_RED_START/END"). This will allow you to use complex Virtual Machines for your most sensitive code and a lighter Virtual Machine for code that needs to be virtualized and executed at a higher speed.

3) From time to time consider updating your virtualization macros to point to a different or newer Virtual Machine architecture. If you have been using the TIGER architecture for some time, you might want to select a different architecture in new versions of your application, to fight against crackers that have been behind your application for some time.

1.3.5 Trial settings



Trial Settings

☑ This application(s) will run in trial/restricted mode, until registered

Trial Restrictions

☑ Days Expiration 30 days

☐ Date Expiration

☑ Executions 40 times

Trial Extension

☑ Allow trial period to be extended (with a valid key)

Trial Extension Manager

Maximum extensions allowed 3

☐ Extension file name extendkey.dat

☐ Extension Registry location HKEY_LOCAL_MACHINE \Software\Company\Product

Value Name trial_ext

Miscellaneous

☑ Run Time (executions) 25 minutes

Clock Change ALLOW_ONE_HOUR_BACK

☐ Global Time 100 minutes

Lock to Country Worldwide (no restriction)

☐ Unlimited (for custom restrictions)

The Trial Settings panel allows you to select different restrictions when your application runs in trial mode. WinLicense offers a wide range of combinations to limit the trial period in your protected application. As opposed to other licensing systems, WinLicense allows you to include several types of restrictions in a single application.

Trial Restrictions

The following trial restriction options can be included before protecting an application:

- **Days Expiration**: This option sets a trial period to a certain amount of days, which is calculated from the day when an application is started on a computer for the first time. Users will not be able to run the application after its trial period has expired, until a trial extension or license key is present.

- **Date Expiration**: This option sets an exact date for the end of the trial period. After this date the user will not be able to run the application, until a trial extension or license key is present.

- **Executions**: This option sets an exact number of executions allowed for an application. Each time the user executes the protected application, the internal counter decreases by one. When the counter reaches zero, the user will not be able to run the application, until a trial extension or license key is present.

Trial Extension

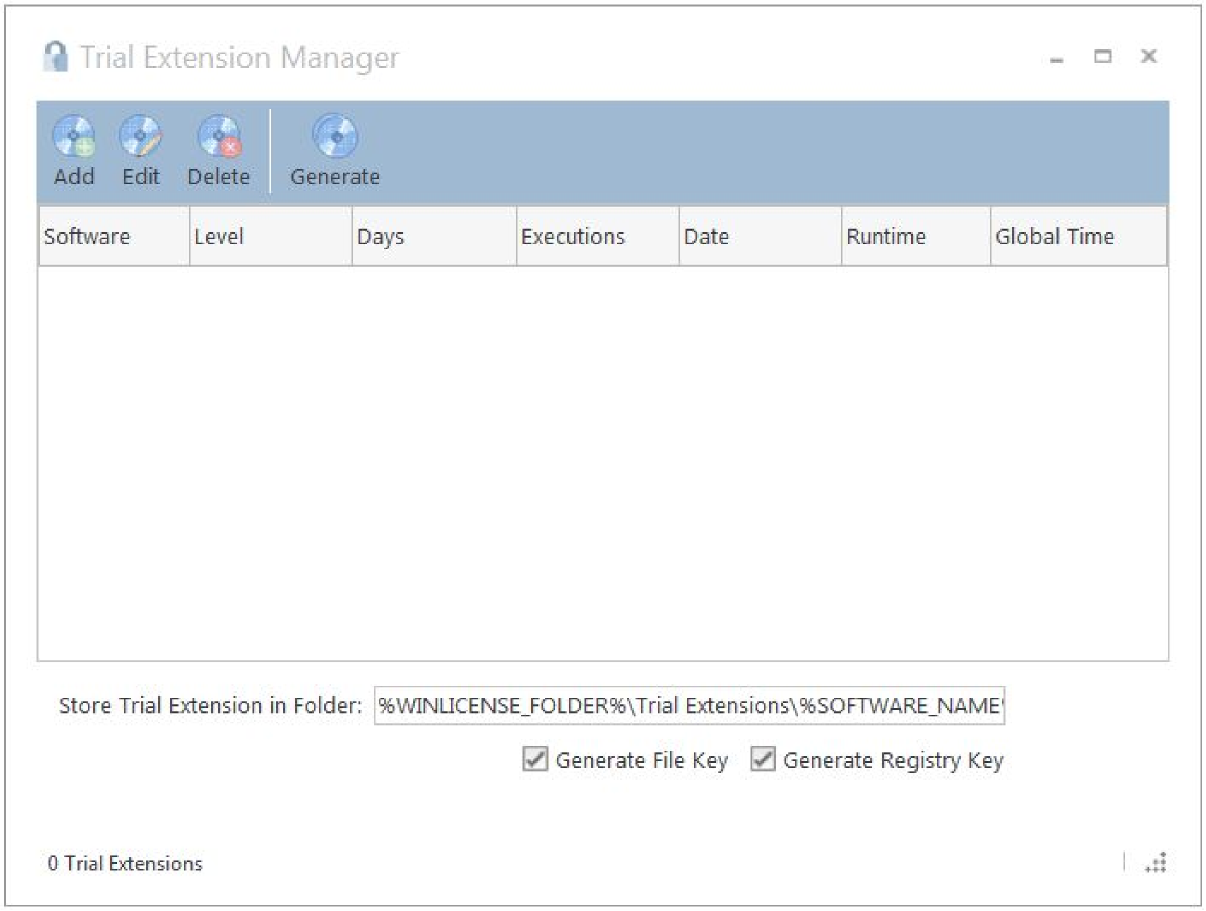
The Trial Extension Manager allows users to extend the trial period of an application with special trial extension keys. If the trial extension option is not enabled, there is not way to extend the trial period in an application after being expired.

A trial extension key can be delivered a file (**Extension file name** option) or as a Windows Registry key (**Extension Registry location** option)

The **Maximum extensions allowed** option limits the amount of trial extensions that an application can be extended. Every time an user adds a new extension key, WinLicense will keep track of it. Users will not be able to extend the trial period when an application has been extended for the specified number of times.

NOTE: If an application has been protected with only some types of expiration, then only the chosen restrictions will be able to be extended with a trial extension key. For example, if only

"Executions" has been set up as a trial limitation, only extension keys with an Executions extension will be able to extend the current trial state.



Software	Level	Days	Executions	Date	Runtime	Global Time

Store Trial Extension in Folder: %WINLICENSE_FOLDER%\Trial Extensions\%SOFTWARE_NAME

✓ Generate File Key ✓ Generate Registry Key

0 Trial Extensions

When generating the trial extension keys, you can use any of the predefined folder constants like: **%WINLICENSE_FOLDER%**, **%INPUT_FILE_FOLDER%**, **%OUTPUT_FILE_FOLDER%, %SOFTWARE_NAME%, %TRIAL_EXTENSION_LEVEL%, %TRIAL_EXTENSION_DAYS%, %TRIAL_EXTENSION_EXECUTIONS%, %TRIAL_EXTENSION_RUNTIME%, %TRIAL_EXTENSION_GLOBAL_TIME%.** Example:

%WINLICENSE_FOLDER%\Trial Extensions\%SOFTWARE_NAME%\%TRIAL_EXTENSION_LEVEL%

Miscellaneous

- **Run Time**: This option sets an exact amount of time (in minutes) that an application can stay in memory. WinLicense will display the Customized Dialog 43 *MSG_ID_TRIAL_RUNTIME_EXPIRED* when the time is over. The internal timer will reset on application restart.

- **Global time**: This option sets an exact amount of total time (in minutes) that an application can run during all Windows® sessions. The internal timer will not reset on restart of operating system. When the global time expires, the application will not be able to run again, until a trial extension or license key is present.

- **Unlimited**: This option allows an application to run in trial mode without any restrictions added by WinLicense. This option is intended to be used to disable internal functionality in the protected application when it runs in trial mode.

- **Lock to Country**: This option restrict the execution of your application (in trial mode) to a specific country.

- **Clock Change:** WinLicense can detect clock back changes to prevent your application from being used indefinitely. In order to give more flexibility to your customers to adapt to clock changes and different time zones when traveling, you can specify the margin of change that you allow putting the clock back:

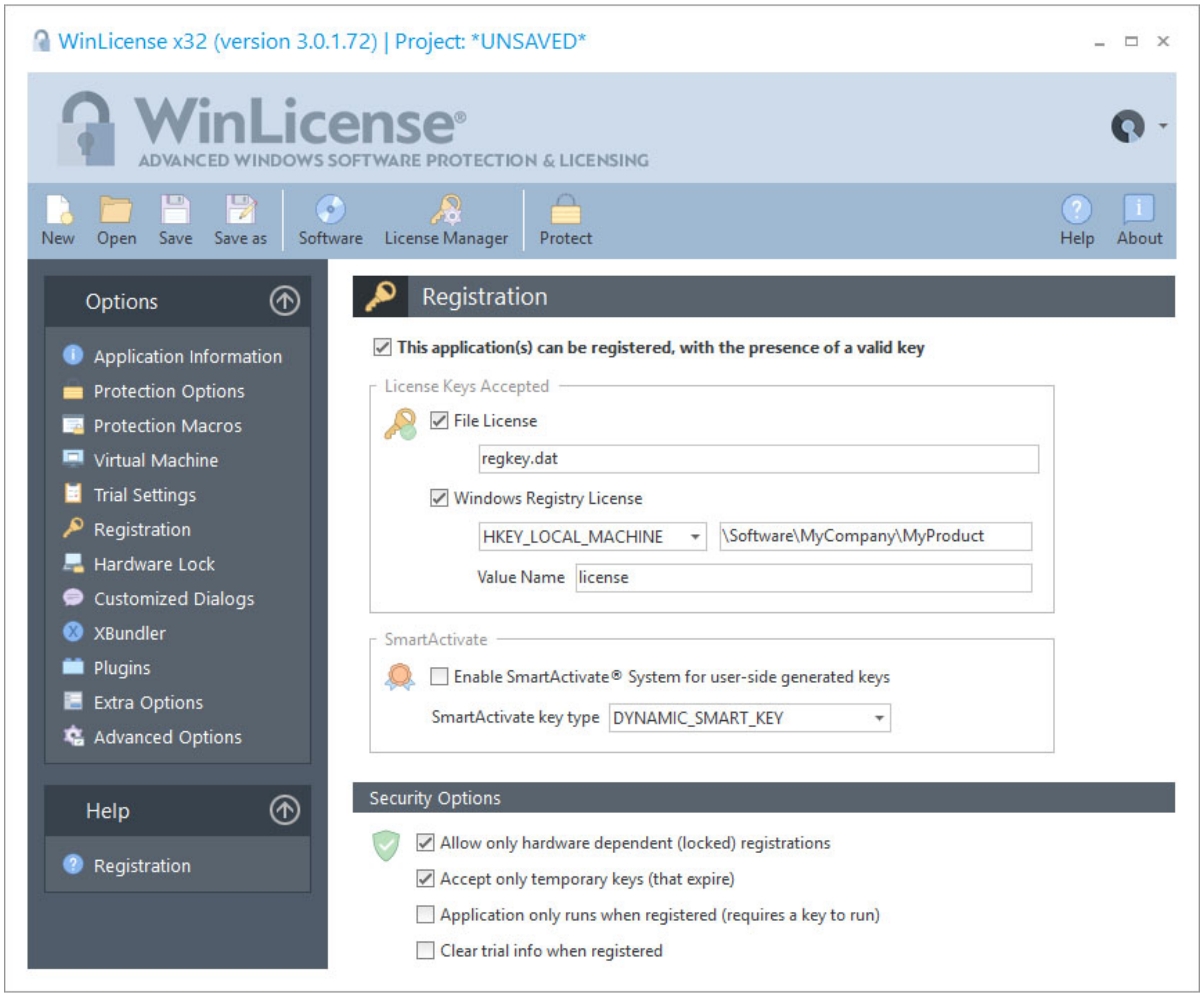
    - **DETECT_ANY_CHANGES**: WinLicense will detect any clock back change. This is the more restricted option to avoid putting the clock back.

    - **ALLOW_ONE_HOUR_BACK**: The user can put the clock back for one hour and WinLicense will not detect the change. This is the recommended option for most situations.

    - **ALLOW_ONE_DAY_BACK**: The user can put the clock back for one day and WinLicense will not detect the change.

    - **ALLOW_ANY_CHANGES**: The user can put the clock back and WinLicense will not detect the change. Notice that if the trial is already expired and the user puts the clock back, the trial will continue as expired because the application is in "trial expired mode".

1.3.6 Registration


WinLicense x32 (version 3.0.1.72) | Project: *UNSAVED*

Registration

☑ This application(s) can be registered, with the presence of a valid key

License Keys Accepted

☑ File License: regkey.dat

☑ Windows Registry License: HKEY_LOCAL_MACHINE \Software\MyCompany\MyProduct

Value Name: license

SmartActivate

☐ Enable SmartActivate® System for user-side generated keys

SmartActivate key type: DYNAMIC_SMART_KEY

Security Options

☑ Allow only hardware dependent (locked) registrations

☑ Accept only temporary keys (that expire)

☐ Application only runs when registered (requires a key to run)

☐ Clear trial info when registered

The Registration panel allows an application to be registered. This means that no trial limitations, which were set on the Trial settings 32 panel, will affect the execution of the registered application. To avoid an application being registered in any way, this panel should be skipped.

WinLicense license keys

WinLicense accepts different types of license keys in order to satisfy different user's needs. Please, refer to the section Licensing with WinLicense 93 for more information.

You can select different types of licensing for a single application. The following types of licenses can be used to register a protected application:

- **File licenses**: To allow single file license keys you have to check the **File License** option and enter the name of the file that will hold the license information when an application is going to be registered. When a user executes a protected application, WinLicense will search for the specified file name in order to registered the application. If the specific license file is found, it will be checked to see if it is a valid license file. You can specify one of the defined WinLicense directory constants to place your licenses in different Windows common folders, like:

    - **%USER_DOCS%** : Specifies the current user documents folder (!My Documents)

    - **%PUBLIC_DOCS%** : Specifies the public documents folder (!All Users\Documents)

    - **%USER_APP_DATA%** : Specifies the current user application data folder (!{user name}\Application Data)

    - **%COMMON_APP_DATA%** : Specifies the common application data for all users (!All Users\Application Data)

    - **%LOCAL_APP_DATA%** : Specifies the local application data for the current user (!{user name}\Local Settings\Application Data (non roaming))

    - **%TEMP_FOLDER%**: Specifies the user temporal folder

    - ***%any_environment_variable%***: Specifies a location defined in an environment variable in runtime

  Notice that the above constanst are case sensitive. You can specify subdirectories with any of the above constants. Example: %USER_DOCS%\MyApplication\Licences\license.dat

  You can also specify a dynamic license key name based on the name of your Software. The string **%SOFT_NAME%** in the license name is replaced by the current Software name (This is useful when you reuse a single project file for several software titles and each one is registered with its own specific license key).

- **Registry licenses**: If you want to register your application using a Windows® Registry key, you have to check the **Windows Registry License** option. Users will have to double-click on the received ".reg" file and agree to import the key in the Windows® Registry. You have to enter the Registry information where your license information will be stored. WinLicense will search the selected Registry information to find the registration data.

- **SmartActivate® licenses**: If you want to include professional and elegant licensing control into your application, you have to check the **Enable SmartActivate® System for user-side generated keys** option. The SmartActivate® System allows an application to be licensed using a valid activation code that can be inserted in a dedicated form in your application. To learn more about the SmartActivate® System, refer to the section below.

- **Text keys**: Text keys are in fact file keys in ASCII format to allow licensing an application from a custom form. WinLicense exports several functions to validate and install a text key in the system in order to register an application. The following functions are used with text keys: WLRegNormalKeyCheck 169, WLRegNormalKeyInstallToFile 172, WLRegNormalKeyInstallToRegistry 174.

The SmartActivate® System

For professional and elegant licensing control, WinLicense offers the SmartActivate® System. This technology allows an application to be registered using SmartActivate® keys, which are composed of the following fields:

- Registration Name, Company and Custom data.

- SmartKey string (I.e, "F5D80C4E-AF3C3B88-D1D2ACF0-EB46BD91-96E29D36-9E35C4DC-75DC")

The SmartKey string will contain information related to license expiration, Hardware ID (specifies in which computer the license can be accepted) and checksums that validate the Registration information (Name, Company and Custom data).

The disadvantages of using SmartActivate® keys is the requirement to create a special form in your application to enter the SmartActivate® information that can read and install (to a file or registry).

When the SmartActivate key system is enabled, you can select between Static or Dynamic 96 SmartActivate keys, so, your application will be ready to accept a SmartActivate key type or another. We recommend just selecting Dynamic SmartActivate keys, as they offer more security and flexibility than Static SmartActivate keys.

Notice that basically a SmartKey is just a "bridge" to finally generate a File or Registry license. You can use the functions WLRegSmartKeyCheck 178 and WLRegSmartKeyInstallToFile 182 /WLRegSmartKeyInstallToRegistry 189 in order to convert the SmartKey into a final File or Registry License. For more information, please, refer to the section Licensing with WinLicense 93.

Security Options

WinLicense offers a set of security options to allow developers to control how an application can be registered and actions to take after that. The different security options are the follows:

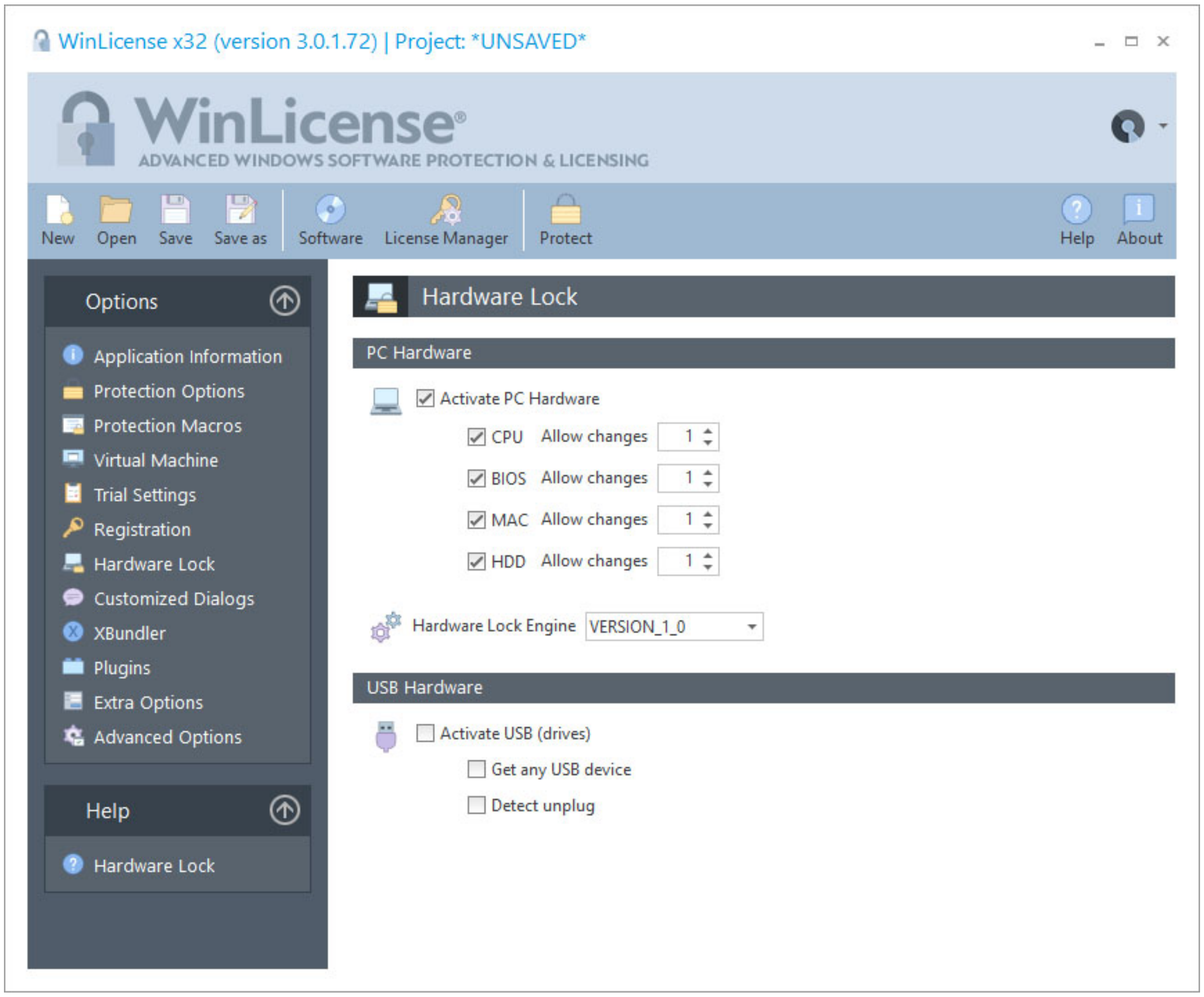
- **Allow only hardware dependent (locked) registrations**: This option restricts the use of registration keys to a specific computer. As each computer has an unique machine ID (processor type, hard disk and BIOS serial), WinLicense can lock a registration key to that specific machine. Please, refer to Hardware Lock 40 panel to get additional information.

- **Accept only temporary keys (that expire)**: This option denies the use of "time unlimited" registration keys and forces WinLicense to accept only temporary registration keys. This option is used as a protection against a developer accidentally releasing a "time unlimited" key for his application.

- **Application only runs when registered (requires a key to run)**: This option allows user to run an application only in the presence of a valid registration key. WinLicense will terminate the application if no key is found or if it is invalid. This option is intended to be used by developers that only allow the execution of their application in registered mode.

- **Clear trial info when registered**: The purpose of this option is to clear the trial information from a computer once an user receives a valid registration key and registers an application. If the user decides to remove the registration key or it expires, the trial period will restart from zero.

1.3.7 Hardware Lock



Hardware Lock

PC Hardware

☑ Activate PC Hardware
☑ CPU Allow changes 1
☑ BIOS Allow changes 1
☑ MAC Allow changes 1
☑ HDD Allow changes 1

Hardware Lock Engine VERSION_1_0

USB Hardware

☐ Activate USB (drives)
☐ Get any USB device
☐ Detect unplug

The Hardware Lock panel is designed to prevent the use of a single registration key with different computers. Even if a registration key is leaked or stolen, that key will not work on other computers. In this panel you can select the different parameters that WinLicense will use to lock machine dependent keys to a computer.

NOTE: To get the Machine ID of your customers, you will have to send them a special application that retrieves their Hardware ID. This Machine ID will be set in a specific field (HardwareId field) in the generated license key 391. You could also customized all this processing by making a special server that communicates with your application in order to retrieve your customer's Machine ID transparently. The function WLHardwareGetId 248 allows you to retrieve the current hardware ID in a specific machine.

PC Hardware

The Machine ID for a specific computer can be taken from several hardware items. WinLicense allows you to select which hardware items will be included to create the final Machine ID for a specific computer. The following hardware items can be included to create the final Machine ID for a specific computer:

- **CPU**: This option uses the CPU features for the current computer. Note that the CPU features are the same for all computers with the same CPU. It is not a good idea to use only this option to generate the final Machine ID.

- **BIOS**: This option uses the BIOS serial number for the current computer.

- **MAC Address**: This option uses the MAC address for the current computer. MAC address should be unique among computers. Be careful including this option if your customers usually change their network cards.

- **HDD Serial**: This option uses the primary hard drive serial number for the current computer.

When a machine dependent license key is going to be registered in the system, the Machine ID in the license key must match the current Machine ID. Once the license is validated and registered, hardware changes will be possible. To allow your customers to change their hardware components, you can select how many hardware changes are allowed for each hardware item. Each time a user replaces a hardware item, its internal counter is decreased by one. When one of these counters reach zero and a new hardware change happens, WinLicense will display the Customized Dialog *MSG_ID_LICENSE_NO_MORE_HW_CHANGES* to indicate that no more hardware changes are allowed.

Hardware Lock Engine

The option Hardware Lock Engine allows you to select a different version of the code that retrieves the PC Hardware ID. In order to keep compatibility with previous WinLicense versions (version <= 2.x) you should select "**VERSION_1_0**". If you are protecting a new application that you have not generated locked licenses for it yet, you might want to select "**VERSION_2_0**". Notice that hardware ID retrieved with VERSION_1_0 is different from the one obtained with VERSION_2_0.

USB Hardware

WinLicense can also lock licenses to a specific USB drive. When a license is locked to a USB drive, WinLicense requires that the USB drive is present when the protected application is launched. If the customer wants to move to a new or different computer, he just needs to plug the USB drive into the new computer. In order to lock a license to a USB drive, you first need to get the USB drive ID from your customer's PC. These are the basic steps to accomplish this task:

1. From within your application, call the function WLHardwareGetNumberUsbDrives to get the number of available USB drives connected to a computer

2. Collect all the USB drives names and IDs and display them to your customer. Call WLHardwareGetIdAt and WLHardwareGetNameAt to retrieve the name and ID of a specific USB drive

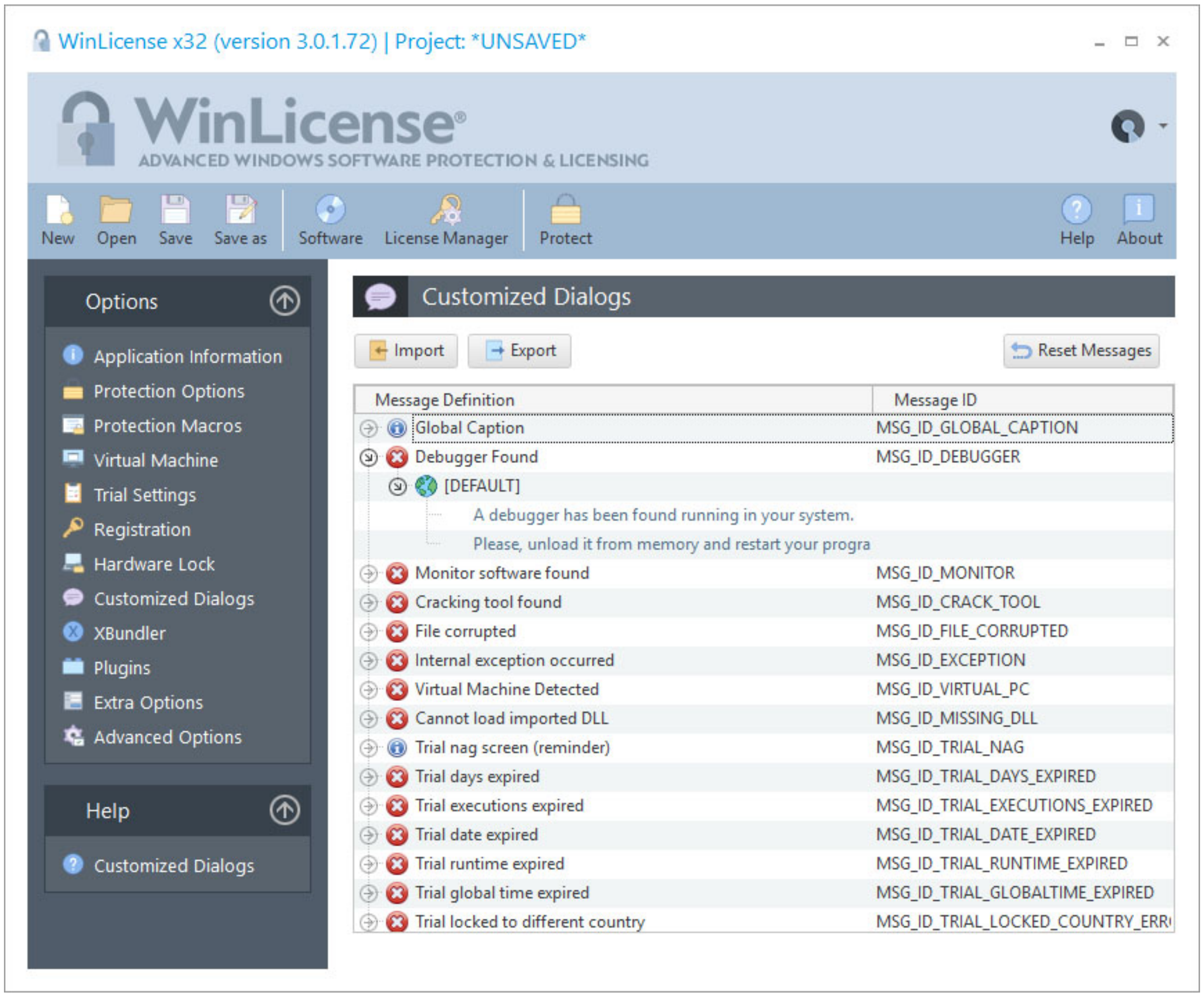
3. From the list that you display to your customer, your customer can recognize (by the USB drive name) the specific USB drive that he wants to use for the hardware locking. He sends you the USB drive ID and you generate a license by filling the "Hardware ID" field with the given USB ID

The option **Get any USB device** will scan for not only USB drives, but any USB device connected to the computer. Notice that not all USB devices contain a serial number or the serial number could be the same for different USB devices.

The option **Detect Unplug** periodically checks (about once each 30 seconds) if the USB drive is connected. If it's disconnected, WinLicense will signal the event (message) **MSG_ID_USB_LOCKING_UNPLUGGED** (from the Customized Dialogs 43 panel).

NOTE: Some old/cheap USB drives do not have an embedded serial number on it. In that case WinLicense will report the ID "0000-0000-0000-0000-0000-0000-0000-FFFF" for those USB drives. You should not lock any license to that hardware ID ("0000-0000-0000-0000-0000-0000-0000-FFFF")

1.3.8 Customized Dialogs



Message Definition	Message ID
Global Caption	MSG_ID_GLOBAL_CAPTION
Debugger Found	MSG_ID_DEBUGGER
[DEFAULT]	
A debugger has been found running in your system.	
Please, unload it from memory and restart your progra	
Monitor software found	MSG_ID_MONITOR
Cracking tool found	MSG_ID_CRACK_TOOL
File corrupted	MSG_ID_FILE_CORRUPTED
Internal exception occurred	MSG_ID_EXCEPTION
Virtual Machine Detected	MSG_ID_VIRTUAL_PC
Cannot load imported DLL	MSG_ID_MISSING_DLL
Trial nag screen (reminder)	MSG_ID_TRIAL_NAG
Trial days expired	MSG_ID_TRIAL_DAYS_EXPIRED
Trial executions expired	MSG_ID_TRIAL_EXECUTIONS_EXPIRED
Trial date expired	MSG_ID_TRIAL_DATE_EXPIRED
Trial runtime expired	MSG_ID_TRIAL_RUNTIME_EXPIRED
Trial global time expired	MSG_ID_TRIAL_GLOBALTIME_EXPIRED
Trial locked to different country	MSG_ID_TRIAL_LOCKED_COUNTRY_ERR

The Customized Dialog window allows changing the messages that may be shown by SecureEngine® when certain situations occur.

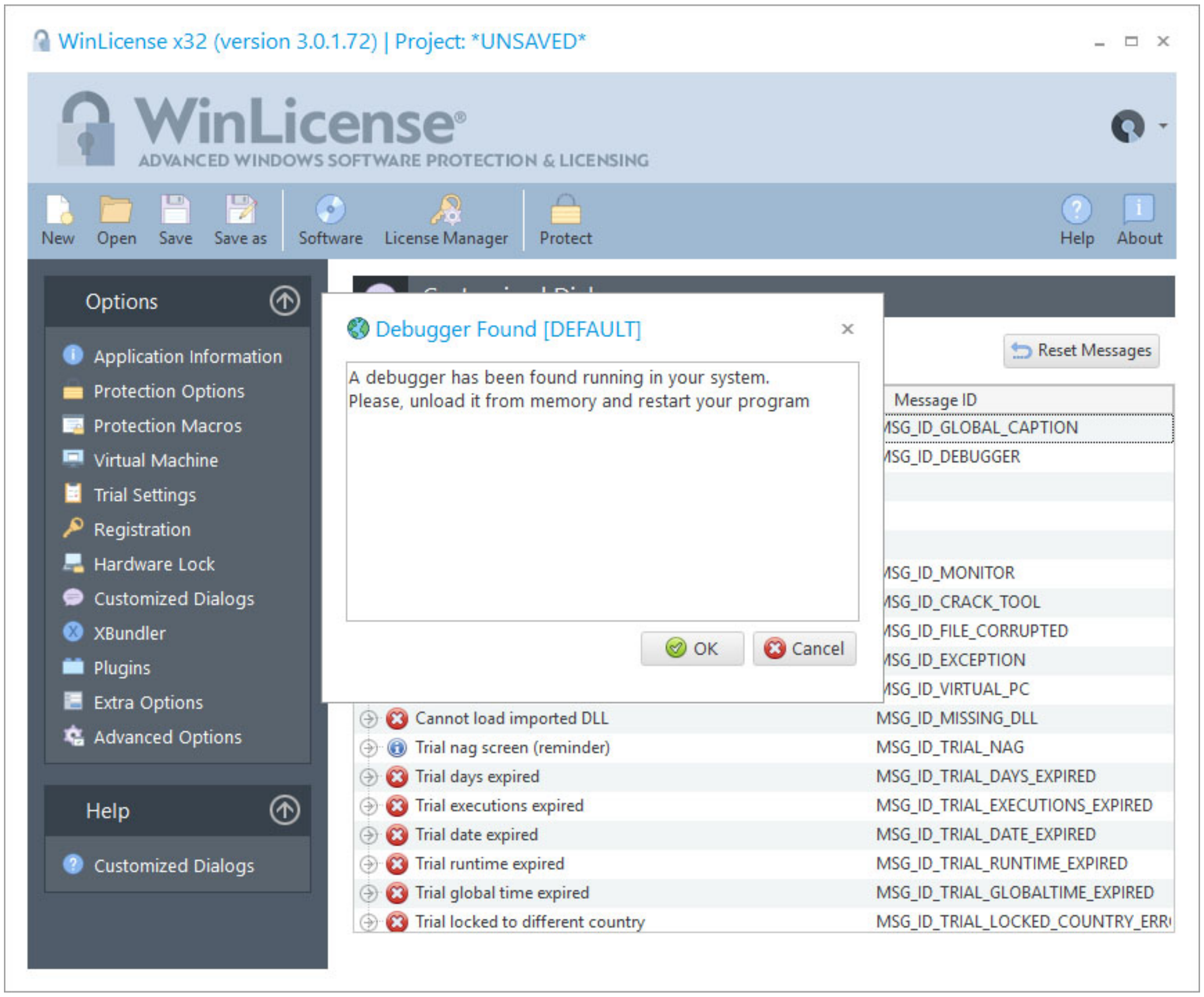
Languages

You can add several languages to each specific message, so the message will be displayed in the user's system language. In order to add a new language for a specific message, just right click on the desired message and select **Add Language**.

The **[DEFAULT]** language is the one that will be displayed in case that the current system language does not match with any of the defined languages for the current message to display. For example, you edit the MSG_ID_DEBUGGER message and define the **[DEFAULT]** and **[GERMAN]** message. If the application is running in a PC with Windows language set to German, the message will be displayed in German (as you defined for **[GERMAN]**). If another customer is running your protected application on a PC with Windows language set to Spanish, the message will be displayed as defined in **[DEFAULT]**

Changing a custom message

To change a custom message, double-click on a specific message to start editing.

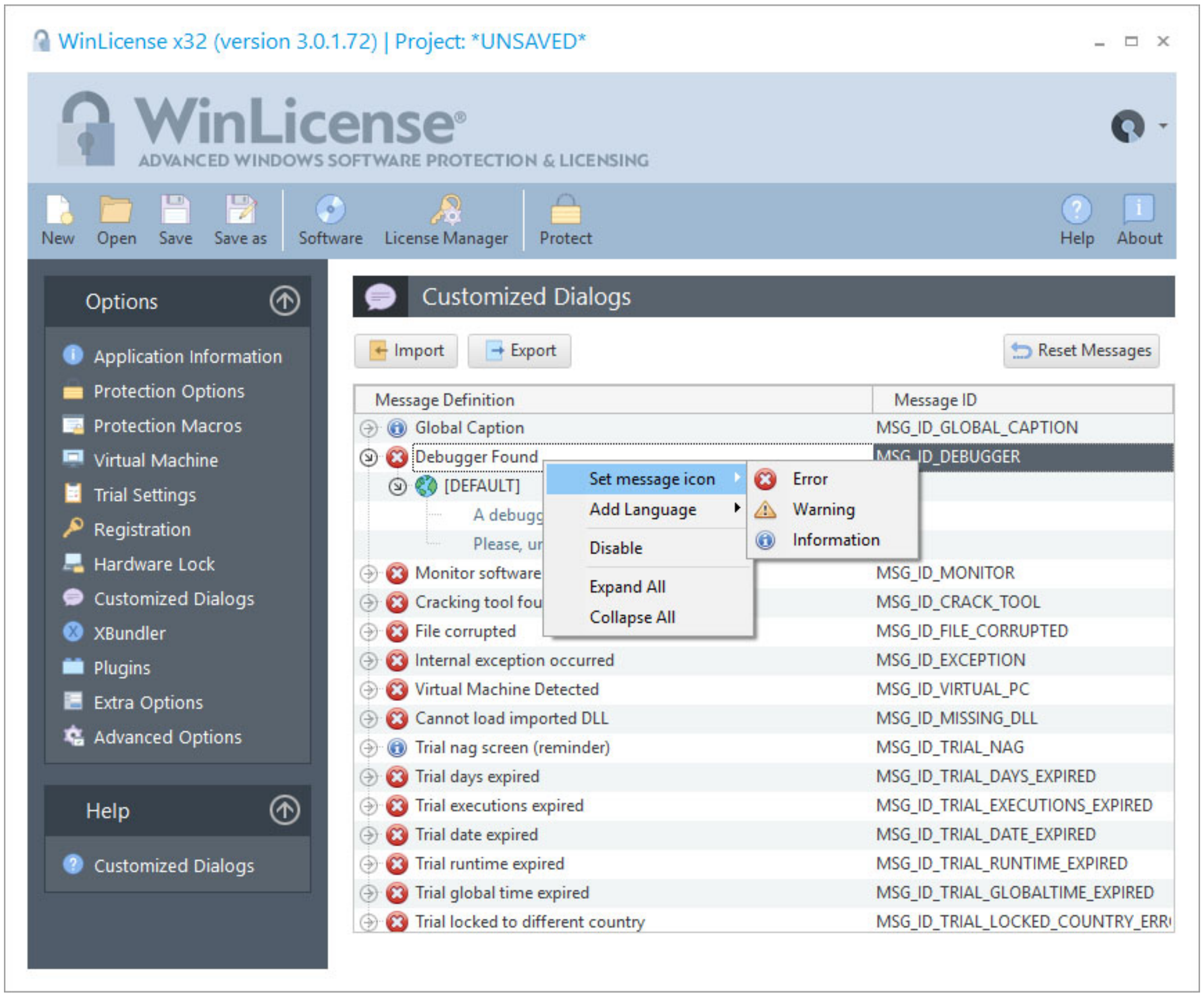


A debugger has been found running in your system.
Please, unload it from memory and restart your program

	Message ID
	MSG_ID_GLOBAL_CAPTION
	MSG_ID_DEBUGGER
	MSG_ID_MONITOR
	MSG_ID_CRACK_TOOL
	MSG_ID_FILE_CORRUPTED
	MSG_ID_EXCEPTION
	MSG_ID_VIRTUAL_PC
Cannot load imported DLL	MSG_ID_MISSING_DLL
Trial nag screen (reminder)	MSG_ID_TRIAL_NAG
Trial days expired	MSG_ID_TRIAL_DAYS_EXPIRED
Trial executions expired	MSG_ID_TRIAL_EXECUTIONS_EXPIRED
Trial date expired	MSG_ID_TRIAL_DATE_EXPIRED
Trial runtime expired	MSG_ID_TRIAL_RUNTIME_EXPIRED
Trial global time expired	MSG_ID_TRIAL_GLOBALTIME_EXPIRED
Trial locked to different country	MSG_ID_TRIAL_LOCKED_COUNTRY_ERR

Changing the custom message icon

Each message is displayed by default as a Windows MessageBox. You can change the MessageBox icon for each message by selecting the specific message and right-click on it, after that use the option **Set Message Icon**.



Message Definition	Message ID
Global Caption	MSG_ID_GLOBAL_CAPTION
Debugger Found	MSG_ID_DEBUGGER
[DEFAULT]	
A debugg	
Please, ur	
Monitor software	MSG_ID_MONITOR
Cracking tool fou	MSG_ID_CRACK_TOOL
File corrupted	MSG_ID_FILE_CORRUPTED
Internal exception occurred	MSG_ID_EXCEPTION
Virtual Machine Detected	MSG_ID_VIRTUAL_PC
Cannot load imported DLL	MSG_ID_MISSING_DLL
Trial nag screen (reminder)	MSG_ID_TRIAL_NAG
Trial days expired	MSG_ID_TRIAL_DAYS_EXPIRED
Trial executions expired	MSG_ID_TRIAL_EXECUTIONS_EXPIRED
Trial date expired	MSG_ID_TRIAL_DATE_EXPIRED
Trial runtime expired	MSG_ID_TRIAL_RUNTIME_EXPIRED
Trial global time expired	MSG_ID_TRIAL_GLOBALTIME_EXPIRED
Trial locked to different country	MSG_ID_TRIAL_LOCKED_COUNTRY_ERR

Changing the general caption for all messages

All messages are shown with a general caption that can also be customized. The general caption message is the **MSG_ID_GLOBAL_CAPTION**. This caption message can be customized in the same way as the rest of the messages, by double clicking it.

Disabling a message from being displayed

The different customized dialogs can be treated as events. When a specific event occurs (like "a debugger found", "trial expired", etc), WinLicense displays the message for the specific event. If you don't want that WinLicense displays a message (do not handle the event) you can disable specific events (messages) from not being handled. For example, if you set the MSG_ID_TRIAL_DAYS_EXPIRED message as disabled, WinLicense will not handle the days expiration event, so the message will not be displayed and your application will keep running as "normal". You are responsible of taking care of that event in runtime (for example by calling WLTrialGetStatus 126) and check if the application is expired, so you can perform your desired actions from within your application.

We recommend that you carefully select which messages are going to be handled by you and make sure that you call the WinLicense SDK to check when that specific event occurs in your application to do the actions that you consider. For example, if you set the MSG_ID_LICENSE_STOLEN as disabled and from within your application you don't call the WinLicense SDK to know the registration status (WLRegGetStatus 161), your application will keep running with the stolen license.

The plugin system in SecureEngine can also be used to handle each specific event (message), so you perform your desired actions from your plugin DLL when a specific event occurs. Please, refer to the Plugins section for more information.

Importing and exporting your customized messages

WinLicense offers the possibility to export your customized messages to a single file so they can be imported in other project files without the need of retype all the messages again. To export all your messages to a file you just need to press the **Export** button and select the name of the file in which your messages will be saved. When you want to import your customized messages to the current project file, just press the **Import** button and select where the file with all your customized messages is located.

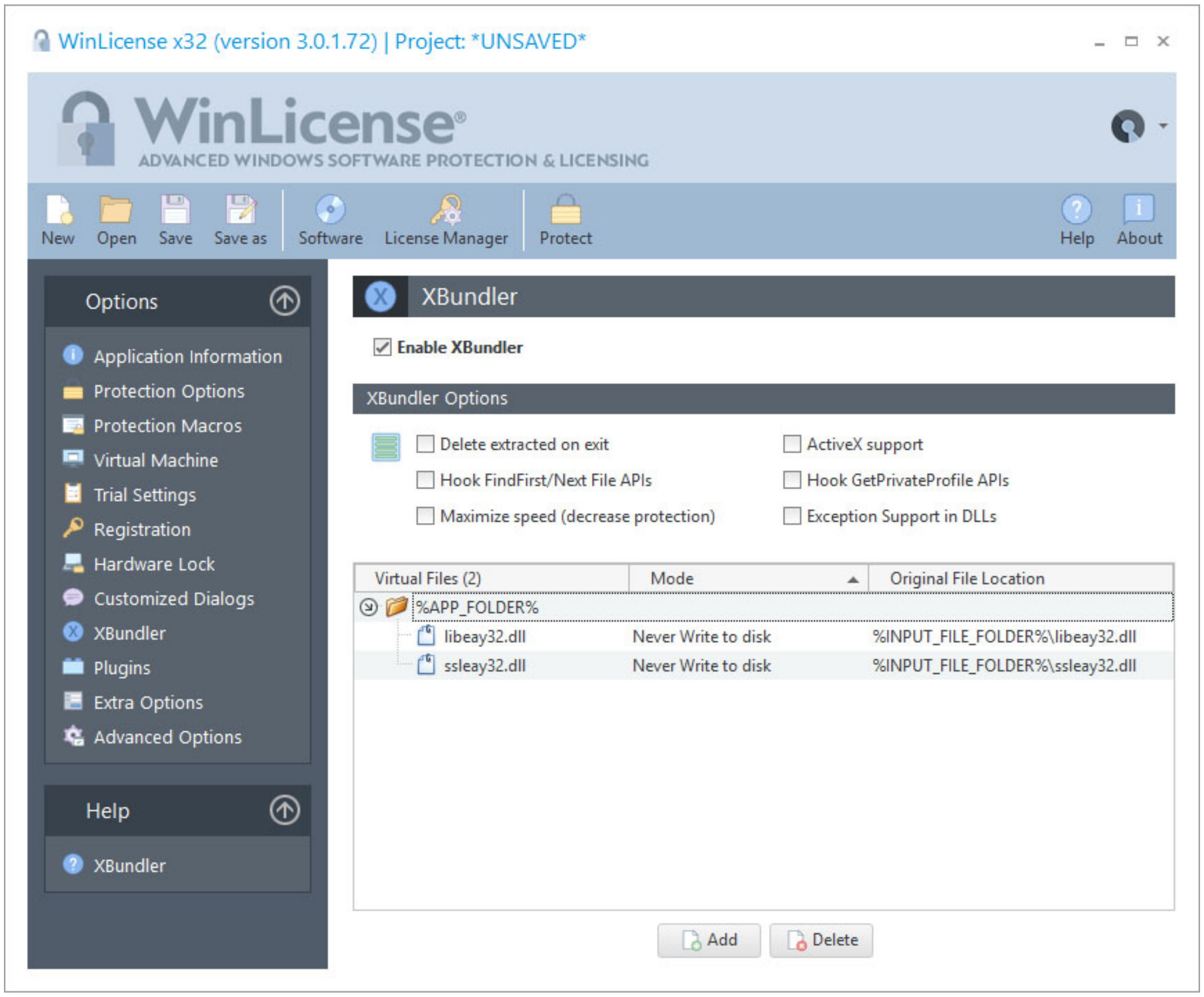
Variables in WinLicense messages

In all trial/registration messages you can insert the following variables that will be replaced with corresponding values:

Variable Name	Description
%daysleft	Number of days left for current trial period.
%execleft	Number of executions left for current trial period.
%totaldays	Total days for current trial period.
%totalexec	Total executions for current trial period.
%expdate	Expiration date for current trial period. Displayed date format is "dd/mm/yyyy".
%name	Registered user's name for current license key.
%company	Registered user's company for current license key.
%machineid	Current Hardware ID for current machine.

NOTE: All variables must be in **lowercase**.

1.3.9 XBundler



[✓] Enable XBundler

[ ] Delete extracted on exit
[ ] Hook FindFirst/Next File APIs
[ ] Maximize speed (decrease protection)
[ ] ActiveX support
[ ] Hook GetPrivateProfile APIs
[ ] Exception Support in DLLs

Virtual Files (2)	Mode	Original File Location
%APP_FOLDER%		
libeay32.dll	Never Write to disk	%INPUT_FILE_FOLDER%\libeay32.dll
ssleay32.dll	Never Write to disk	%INPUT_FILE_FOLDER%\ssleay32.dll

XBundler allows you to embed DLLs and data files inside a protected application, simplifying the distribution of your application to your customers and avoiding your DLLs and data files being used by third party software. XBundler compresses and encrypts all of the embedded files without affecting the ability of your application to function correctly and with no additional coding.

When your application wants to access your embedded DLLs and/or data files, XBundler will not write the embedded files to disk. Instead, XBundler uses a special application hooks to detect when an application is accessing embedded DLLs and/or data files and will decrypt/encrypt the required block of data.

Scenarios for using XBundler

XBundler can be used in many scenarios. The most common ones are:

- Protect your DLLs from being reused by third party software: When you select not to write your files to disk, XBundler will keep your files totally encrypted and will access them directly in memory after decrypting the necessary blocks of data. Given that your DLLs are not written to disk, third party software cannot reuse your DLLs for their own benefits.

- Solve "DLL Hell" issues: XBundler will guarantee that your application is always using your embedded DLLs. This will avoid users and applications from modifying/deleting your DLLs hence stopping your application from working.

- Protect your DLLs against reverse engineering: XBundler encrypts your DLL and/or data files to prevent them from being extracted directly from your application. Besides, Themida/WinLicense will seat on top of XBundler supervising the system against any cracking activity, protecting your embedded DLLs and your main application with the latest technology in software protection.

- Compress your DLLs and data files: XBundler will compress all of your embedded DLLs and data files reducing their size by 35-60% and using a very fast decompression algorithm which does not decrease your applications performance.

- Protect your media files: If your application uses exclusive designs with graphics, music, video, etc. XBundler can embed all of these media files with your application to avoid other people directly viewing them, or using them for their own software.

XBundler Files Panel

To add files to be embedded inside your final protected application, you can either drag the file to the XBundler panel or select a file using the **Add** button. The file will appear in the list if it has not already been included.

- The **Virtual File** column displays the location where a specific file can be found in runtime in case that you select the option to extract the file to disk. You can create your own extraction hierarchies by creating subfolders in the Virtual File column. To do so, just right-click on the XBundler files panel and select the option "Add Folder". If you want to change the root folder for the virtual file, select the option "Add Root Folder". The current defined values are:

  - **%APP_FOLDER%**: This is the folder from where you protected application is executed
  - **%WIN_FOLDER%**: Windows folder
  - **%WINSYS_FOLDER%**: Windows System folder
  - **%USER_DOCS%**: Current user documents folder
  - **%LOCAL_APP_DATA%**: Current user local AppData folder
  - **%COMMON_APP_DATA%**: Common application data for all users

- The **Mode** column allows you to select if the file will be extracted to disk in runtime or the file will never be extracted to disk. When the file is not extracted to disk, XBundler uses process hooking in order to detect file accesses and redirect them to specific locations within the process space. If you want to extract the file to disk, there are several types of extraction options to suit different developer needs.

- The **Original File Location** column specific the location of the file on disk. This is used in protection time in order to read the file to embed. If you don't want to work with full paths, you can use special constants for the file location, like **%WINLICENSE_FOLDER%**, **%INPUT_FILE_FOLDER%**, **%OUTPUT_FILE_FOLDER%**, **%PROJECT_FOLDER%**. Example:

  %INPUT_FILE_FOLDER%\files\my_file.dat

XBundler Options

- **Delete extracted on exit**: In case that for any of your embedded files you have selected the option "Extract to disk", this option will delete the extracted file once that the application exits. If you are selecting the option "Never extract to disk" for all your embedded files, this option has no effect.

- **Hook FindFirst/FindNext File APIs**: This option hooks the FindFirst/FindNext Windows API. These APIs are normally used by Windows when files are going to be listed in a Windows Shell Dialog. If you want to make your embedded files visible to W¡ndows Shell Dialogs or you want to enumerate your embedded files from inside your application (using FindFirstFile, FindNextFile, etc.) you have to select this option. Note that even if you see your embedded files from inside your application when you select this option, your embedded files will NOT be visible to users and other applications.

- **Maximize speed (decrease protection)**: This option will decrease the encryption/virtualization of the XBundler protection code to avoid a performance decrease in case that you access to your embedded files quite frequently.

- **ActiveX support**: This option allows you to register your embedded DLLs/OCXs before your application starts. It has the same effect as if "regsvr32" is performed before your application starts. Notice that in order to register your embedded DLLs/OCXs, your application needs to be running with administrator's rights. If the application is running with restricted user rights, the embedded DLLs won't be register in the system. You have to make sure that the protected application runs with administrator's rights the first time that is executed in the system to allow registration of embedded DLLs.

- **Hook GetPrivateProfile APIs**: This options hooks the Win32 GetPrivateProfile APIs in case that your application use those functions to access to the files that are going to be bundled (and never written to disk). This option should be checked in your are bundling .INI files with the option "Never write to disk".

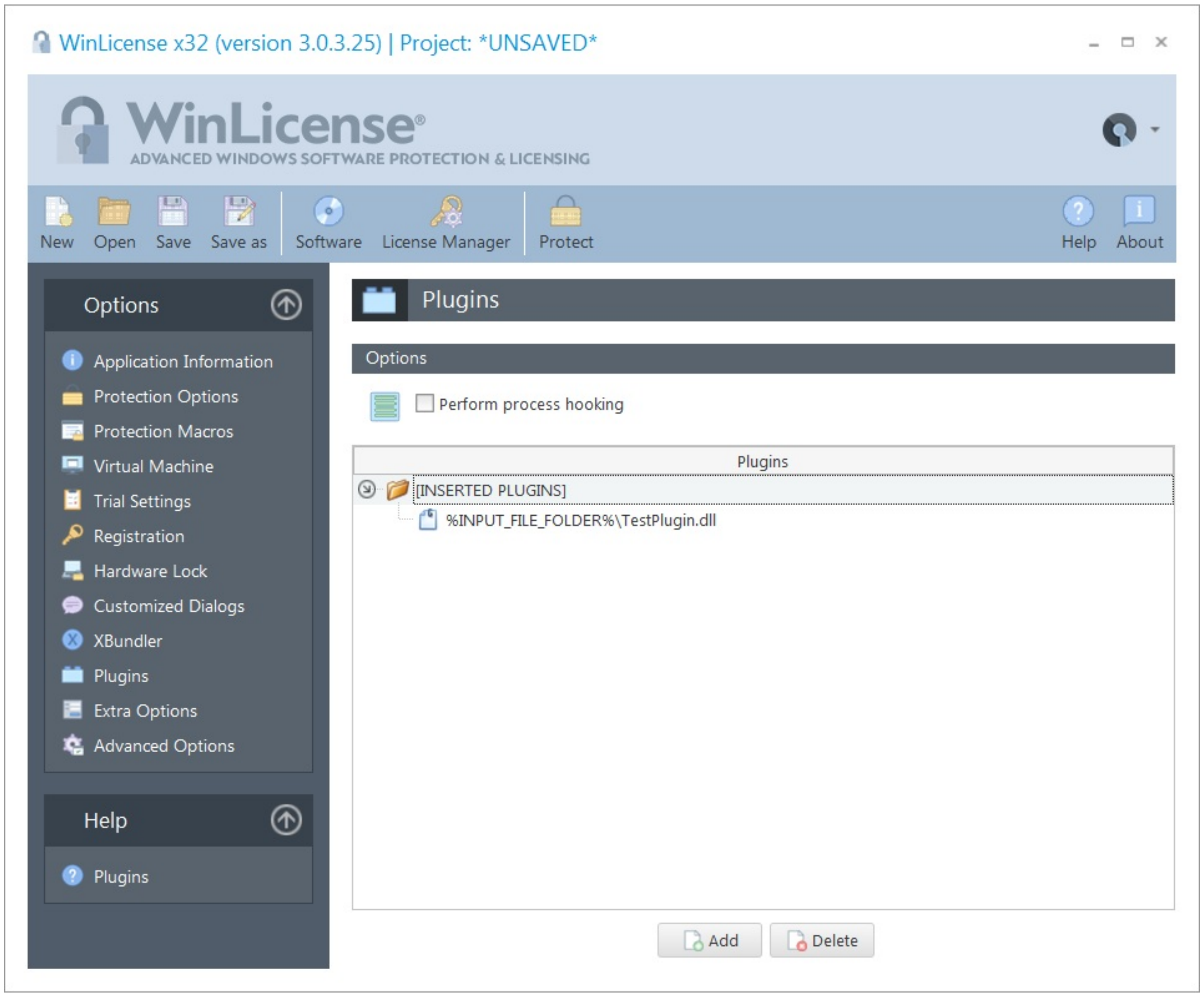
- **Exception support in DLLs**: Some DLLs generates exceptions (handled) on start and that interacts with the exception handling in the protection code. If any of your embedded DLLs produce handled exceptions on start, you have to check this option.

Changing the Extraction Mode for all selected files

If you want to change the extraction type for several files at the same time, just select all wanted files and press:

- CTRL + 0 = "Never Write to disk"
- CTRL + 1 = "Extract always"
- CTRL + 2 = "Extract if not exists"
- CTRL + 3 = "Extract if older exists"
- CTRL + 4 = "Extract if different exists"

1.3.10 Plugins



WinLicense allows you to insert custom plugins (DLLs) that will be embedded inside the protected binary. The embedded plugin can implement specific defined callbacks that will be called when a specific protection event occurs, so you can have more control on the protection, add your own custom protections, etc.

A plugin is basically a compiled native DLL (.NET DLLs are not supported), that exports specific functions names that matches a specific name pattern. For example, the callback "***SecureEngineInitialize***" (notice the *wildcard*) means that you can define any function (to export) that will contain the "SecureEngineInitialize" string within your function name. For example, the function name *MyPlugin_SecureEngineInitialize* will match the "***SecureEngineInitialize***" callback.

Options

- **Perform process hooking**: This option will fully emulate the loading of your DLL in memory. This option is only required for specific plugins. Most plugins will work fine without this option. The preferred is to have this option unchecked, because it won't perform any hooking on the current process.

Testing your plugins

It's a good idea to test your plugins after your modify them, just to make sure that the calling convention (***stdcall***) and parameters are defined as expected. To test your plugin, you can just right-click on it and select "**Test Plugin**". Your defined callbacks will be called with default/dummy parameters to test your callbacks. If a callback fails (produces exception, etc) it will be reported on the User Interface.

Supported Compilers

There are no restrictions about the compiler used to create a plugin. The only requirement is that the plugin cannot be a .NET (or mixed managed) DLL. Only native DLLs are supported.

In case that you are using Visual Studio to create your plugin, you should avoid the explicit linking with the Microsoft Runtime Libraries (such as MSVCR100, etc.). You should compile your DLL with the /MT compiler switch.

Plugin Callbacks

The plugin system will be extended in future versions with new callbacks. The current defined callbacks (name patterns) are:

- SecureEngineInitialize 54
- SecureEngineFinalize 54
- SecureEngineShowCustomMessage 55
- SecureEngineProcessHardwareId 56
- SecureEngineFirstRunTrial 57
- SecureEngineFirstRunRegistered 58
- SecureEngineGetLicenseInfo 59

- SecureEngineGetApplicationStatus 61
- SecureEngineDoRegistration 62
- SecureEngineGetEncryptionKey 63

1.3.10.1 SecureEngineInitialize

This function is called when the protection starts, before your application has been processed (decrypted, decompressed, etc) to be executed in memory. This can be a good place if you want to add your own protection checks, etc.

Show C/C++ function definition

```
STDCALL bool SecureEngineInitialize(void);
```

Show Delphi function definition

```
function SecureEngineInitialize():Boolean; stdcall;
```

Return Values

If your callback returns FALSE, the application will be terminated. If it returns TRUE, the protection will continue execution.

1.3.10.2 SecureEngineFinalize

This function is called when the protection boot loader has been executed, your application is ready to have control of the CPU.

Show C/C++ function definition

```
STDCALL bool SecureEngineFinalize(void);
```

Show Delphi function definition

```
function SecureEngineFinalize():Boolean; stdcall;
```

Return Values

If your callback returns FALSE, the application will be terminated. If it returns TRUE, the protection will continue execution.

1.3.10.3 SecureEngineShowCustomMessage

This function is called when a Customized Dialog 43 is going to be displayed by the protection. This function receives the message that is going to be displayed by the protection in ANSI format (SecureEngineShowMessageA) or UNICODE format (SecureEngineShowMessageW)

Show C/C++ function definition

```
STDCALL bool SecureEngineShowCustomMessageA(
    int      CustomMessageId,
    char*    CustomMessageString
);

STDCALL bool SecureEngineShowCustomMessageW(
    int      CustomMessageId,
    wchar_t* CustomMessageString
);
```

Show Delphi function definition

```
function SecureEngineShowCustomMessageA(
    CustomMessageId:Integer;
    CustomMessageString:PAnsiChar
):Boolean; stdcall;

function SecureEngineShowCustomMessageW(
    CustomMessageId:Integer;
    CustomMessageString:PWideChar
):Boolean; stdcall;
```

Parameters

CustomMessageId

[in] Identifier for the message that is going to be displayed. Please, refer to the *CustomMessagesConstantsDefinitions.h* (for C/C++) or *CustomMessagesConstantsDefinitions.inc* (for Delphi) under the /WinLicenseSDK/ExamplesSDK/Plugins/Include subfolder.

CustomMessageString

[in] Pointer to a null-terminated string with the message that is going to be displayed.

Return Values

If the function handles the message, you should return TRUE, that means that the protection will not display the message.

If the function does not handle the message or you want that the protection proceeds displaying the message, you should return FALSE.

1.3.10.4 SecureEngineProcessHardwareId

This function allows you to change the hardware ID retrieved by WinLicense by your own one, so you can use the licensing system offered by WinLicense but using your own hardware ID. This function receives the hardware ID retrieved by WinLicense in ANSI format (SecureEngineProcessHardwareIdA) or UNICODE format (SecureEngineProcessHardwareIdW).

Show C/C++ function definition

```
STDCALL bool SecureEngineProcessHardwareIdA(
    char*    HardwareId
);

STDCALL bool SecureEngineProcessHardwareIdW(
    wchar_t* HardwareId
);
```

Show Delphi function definition

```
function SecureEngineProcessHardwareIdA(
    HardwareId:PAnsiChar
):Boolean; stdcall;

function SecureEngineProcessHardwareIdW(
    HardwareId:PWideChar
):Boolean; stdcall;
```

Parameters

HardwareId

[in/out] This parameter contains the hardware ID obtained by WinLicense. If you are going to modify the hardware ID by your own one, you have to overwrite this buffer.

Return Values

If the function has changed the hardware ID, you have to return TRUE, the hardware ID will be internally modified by your own one.

If the function is not changing the Hardware Id, you have to return FALSE.

Remarks

The returned hardware ID must have the expected format for the WinLicense hardware ID. The format of the hardware ID is:

```
1111-2222-3333-4444-5555-6666-7777-8888
```

If you want to display a shorted hardware ID for your customers, you can do it with no problems, for example, you show them **AAAA-BBBB**, but internally you have to pass to WinLicense something like:

```
AAAA-BBBB-0000-0000-0000-0000-0000-0000
```

1.3.10.5 SecureEngineFirstRunTrial

This function is called by WinLicense the first time that an application runs in trial mode in the system. This function receives the hardware ID retrieved by WinLicense in ANSI format (SecureEngineFirstRunTrialA) or UNICODE format (SecureEngineFirstRunTrialW). It also receives a pointer to a structure with the trial expiration information (days left, executions left, expiration date and global time left).

Show C/C++ function definition

```
STDCALL bool SecureEngineFirstRunTrialA(
    char*          HardwareId,
    TTrialExpInfo* TrialExpInfo
);

STDCALL bool SecureEngineFirstRunTrialW(
    wchar_t*       HardwareId,
    TTrialExpInfo* TrialExpInfo
);
```

Show Delphi function definition

```
function SecureEngineFirstRunTrialA(
    HardwareId:PAnsiChar;
    TrialExpInfo:TTrialExpInfo
):Boolean; stdcall;

function SecureEngineFirstRunTrialW(
    HardwareId:PWideChar;
    TrialExpInfo:TTrialExpInfo
):Boolean; stdcall;
```

Parameters

HardwareId

[in] This parameter specifies the hardware ID of the current machine.

TrialExpInfo

[in] This parameter specifies a pointer to a TTrialExpInfo structure with information about the current trial expiration (days, executions, date and global time left)

Return Values

If the function returns True, WinLicense will continue running the protected application.

If the function returns False, WinLicense will terminate the protected application immediately. If the application is launched again, the SecureEngineFirstRunTrial function will be called again until it returns True.

1.3.10.6 SecureEngineFirstRunRegistered

This function is called by WinLicense when a license is installed in the system. If you run the application again (with the same license) this function won't be called. If you install a new license (replacing the previous one), this function will be called again with the information of the new license.

Show C/C++ function definition

```
STDCALL bool SecureEngineFirstRunRegisteredA(
    char*       pHardwareId,
    char*       pRegName,
    char*       pCompany,
    char*       pCustomData,
    TRegExpInfo RegExpInfo
);

STDCALL bool SecureEngineFirstRunRegisteredW(
    wchar* pHardwareId,
    wchar* pRegName,
    wchar* pCompany,
    wchar* pCustomData,
    TRegExpInfo RegExpInfo
);
```

Show Delphi function definition

```
function SecureEngineFirstRunRegisteredA(
    pHardwareID:PAnsiChar;
    pRegName:PAnsiChar;
    pCompany:PAnsiChar;
    pCustomData:PAnsiChar;
    RegExpInfo:TRegExpInfo
):Boolean; stdcall;
```

```
function SecureEngineFirstRunRegisteredW(
    pHardwareID:PWideChar;
    pRegName:PWideChar;
    pCompany:PWideChar;
    pCustomData:PWideChar;
    RegExpInfo:TRegExpInfo
):Boolean; stdcall;
```

Parameters

pHardwareId

[in] This parameter specifies the hardware ID of the current machine.

pRegName

[in] This parameter specifies the registration name for the current license key.

pCompany

[in] This parameter specifies the registration company for the current license key.

pCustomData

[in] This parameter specifies the custom data for the current license key.

RegExpInfo

[in] This parameter specifies a pointer to a TRegExpInfo structure with information about the current license key expiration (days, executions, date and global time left)

Return Values

If the function returns True, WinLicense will continue running the protected application.

If the function returns False, WinLicense will terminate the protected application immediately. If the application is launched again, the SecureEngineFirstRunRegistered function will be called again until it returns True.

1.3.10.7 SecureEngineGetLicenseInfo

This function is called by WinLicense when an application is running in registered mode. WinLicense will pass the registration information to this function, such as Registration Name, Company and Custom Data.

Show C/C++ function definition

```
STDCALL bool SecureEngineGetLicenseInfoA(
    char*       pRegName,
    char*       pCompany,
    char*       pCustomData
);

STDCALL bool SecureEngineGetLicenseInfoW(
    wchar* pRegName,
    wchar* pCompany,
    wchar* pCustomData
);
```

Show Delphi function definition

```
function SecureEngineGetLicenseInfoA(
    pRegName:PAnsiChar;
    pCompany:PAnsiChar;
    pCustomData:PAnsiChar
):Boolean; stdcall;

function SecureEngineGetLicenseInfoW(
    pRegName:PWideChar;
    pCompany:PWideChar;
    pCustomData:PWideChar
):Boolean; stdcall;
```

Parameters

pRegName

[in] This parameter specifies the registration name for the current license key.

pCompany

[in] This parameter specifies the registration company for the current license key.

pCustomData

[in] This parameter specifies the custom data for the current license key.

Return Values

If the function returns True, WinLicense will continue running the protected application.

If the function returns False, WinLicense will terminate the protected application immediately.

1.3.10.8 SecureEngineGetApplicationStatus

This function is called by WinLicense to report the current application status. The main status of an application is either Trial and Registered. You can also get extended status information like: Trial Expired, Trial Manipulated, Invalid Hardware ID, Corrupted License Key, etc. through this function.

Show C/C++ function definition

```
STDCALL bool SecureEngineGetApplicationStatus(
    int            ApplicationStatus,
    int            ExtendedStatus,
    TTrialExpInfo* TrialExpInfo,
    TRegExpInfo*   RegExpInfo
);
```

Show Delphi function definition

```
function SecureEngineGetApplicationStatus(
    ApplicationStatus:Integer;
    ExtendedStatus:Integer;
    TrialExpInfo:TrialExpInfo;
    RegExpInfo:TRegExpInfo
):Boolean; stdcall;
```

Parameters

ApplicationStatus

[in] This parameter specifies the status of the current application. The possible status are Trial (1) and Registered (0).

ExtendedStatus

[in] This parameter contains extended information about the current status of an application. The current extended status values are:

- *wdcNoError* (0): The application runs in either Trial or Registered mode with no errors or expiration in trial or registration
- *wdcInvalidLicense* (2): The current installed license is invalid or corrupted
- *wdcInvalidHardwareLicense* (3): The license is locked to another machine
- *wdcNoMoreHwdChanges* (4): No more hardware IDs changes are allowed
- *wdcLicenseExpired* (5): The current license key is expired. See RegExpInfo for more information about expirations

- *wdcInvalidCountryLicense* (6): The current license key is locked to a different country
- *wdcLicenseStolen* (7): The current license key has been marked as stolen
- *wdcWrongLicenseExp* (8): The current license key is a permanent key and only licenses that expire are allowed in the protected application
- *wdcWrongLicenseHardware* (9): The current license key does not have machine ID information (and machine ID is required in the protected application)
- *wdcTrialExpired* (10): The trial period has expired. See TrialExpInfo for more information about expirations
- *wdcTrialManipulated* (11): The trial period has been manipulated by external user/attacker

TrialExpInfo

[in] This parameter specifies a pointer to a TTrialExpInfo structure with information about the current trial expiration (days, executions, date...)

RegExpInfo

[in] This parameter specifies a pointer to a TRegExpInfo structure with information about the current license key expiration (days, executions, date...)

Return Values

If the function returns True, WinLicense will continue running the protected application.

If the function returns False, WinLicense will terminate the protected application immediately.

1.3.10.9 SecureEngineDoRegistration

This function is called by WinLicense to allow an application to be registered before it gets executed. WinLicense will pass an array of functions pointers to allow calling all necessary functions to register an application.

Show C/C++ function definition

```
STDCALL bool SecureEngineDoRegistrationA(
    TWDCfunctionsArray *pArrayFunctions
);
```

```
STDCALL bool SecureEngineDoRegistrationW(
    TWDCfunctionsArrayW *pArrayFunctions
);
```

Show Delphi function definition

```
function SecureEngineDoRegistrationA(
    var pArrayFunctions:TWDCfunctionsArray
):Boolean; stdcall;

function SecureEngineDoRegistrationW(
    var pArrayFunctions:TWDCfunctionsArrayW
):Boolean; stdcall;
```

Parameters

pArrayFunctions

[in] This parameter a specifies an array of functions (TWDCfunctionsArray) with pointers to specific registration functions.

Return Values

If the function returns True, WinLicense will continue running the protected application.

If the function returns False, WinLicense will terminate the protected application immediately.

1.3.10.10 SecureEngineGetEncryptionKey

This function allows you to specify an encryption key to encrypt/decrypt specific areas in your application. At the moment, only the encryption/decryption of the different sections in the PE file is supported (parameter "ZoneId = 0")

This function is called in protection time to get the encryption key from your plugin. The retrieved encryption key will be used to encrypt your application (apart from other encryption layers applied to encrypt your application).

In runtime, the protection will call your embedded plugin to retrieve the encryption key to decrypt your application. Once that the key has been used, the buffer with the retrieved key will be destroyed.

Show C/C++ function definition

```
STDCALL bool SecureEngineGetEncryptionKey(
    int      ZoneId,
    char*    OutputEncryptionString
);
```

Show Delphi function definition

```
function SecureEngineGetEncryptionKey(
    ZoneId:Integer;
    OutputEncryptionString:PAnsiChar
):Boolean; stdcall;
```

Parameters

ZoneId

[in] Identifies the zone for where the encryption key is required. At the moment only 0 (zone 0) is supported, meaning the encryption key to encrypt the application code/data.

OutputEncryptionString

[out] Pointer to an allocated buffer where the encryption key will be copied.

Return Values

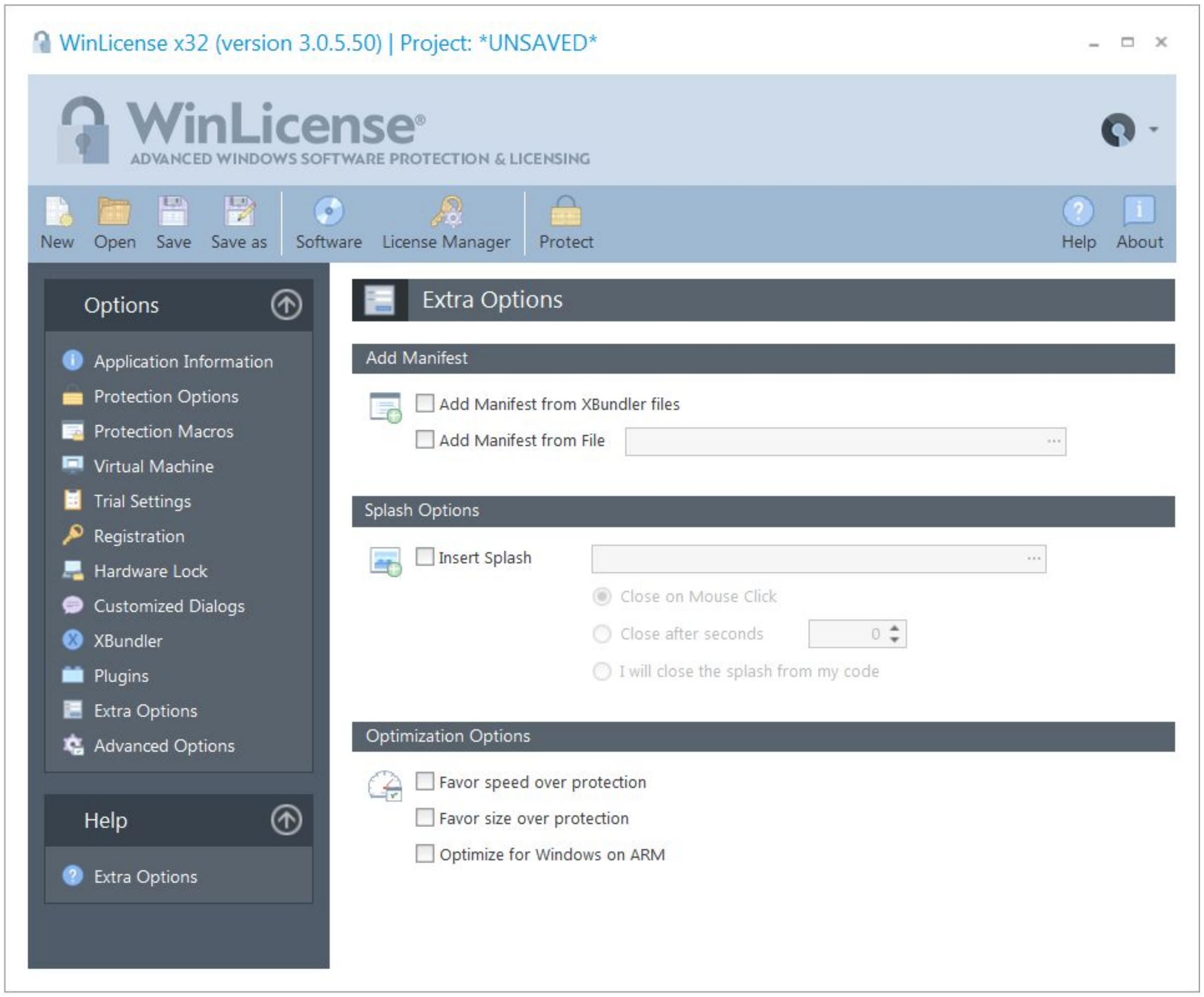
f the function returns the encryption key for the specified zone, you should return TRUE. Otherwise, return FALSE.

Remarks

At the moment, this function is only supported in 32-bit applications.

The returned encryption key must be a NULL terminated ANSI string. The size must be less than 256.

1.3.11 Extra Options



Extra Options

Add Manifest

- [ ] Add Manifest from XBundler files
- [ ] Add Manifest from File

Splash Options

- [ ] Insert Splash
  - (●) Close on Mouse Click
  - ( ) Close after seconds 0
  - ( ) I will close the splash from my code

Optimization Options

- [ ] Favor speed over protection
- [ ] Favor size over protection
- [ ] Optimize for Windows on ARM

Manifest Options

You can explicitly add a custom manifest resource to your protected application in case that it's not present in your unprotected application.

- **Add Manifest from XBundler files**: Sometimes a bundled DLL (not extracted to disk) requires a manifest to operate correctly (for example, it requires admin's rights). If the DLL is bundled, that manifest is not seem by the operating system, so it cannot apply it before the application starts. This option extracts the manifest of the bundled files and apply them to the protected application.

- **Add Manifest from File**: You can add any manifest information into your protected application from an external text file. If you require to extract the manifest from an external binary (EXE/DLL) you can use any PE file editor that can read/display the resources section and you can grab the manifest (in text format) from there. After that, just create a text file with that information and pass the path to that file into the "Add Manifest From File" option. This is an example of the content of a possible external manifest file to require admin's rights in the protected application:

```
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" mani-
festVersion="1.0">
  <assemblyIdentity version="1.0.0.0"
       processorArchitecture="X86"
       name="AliasDatabaseServer"
       type="win32" />
  <description>AlaiasDatabaseServer manifest</description>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level="requireAdminis-
trator" />
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

Splash Options

You can insert a splash image (JPG image) that will be displayed before your application starts (while the protection boot code is running). If you use a heavy virtual machine for the boot protection in the Virtual Machine 30 panel, a splash screen is ideal to let your customers know that your application is booting up.

When the splash is displayed, there are several ways to close the splash screen, like **Close on Mouse Click**, **Close after** a number of **seconds** or close it from the WinLicense SDK (WLSplashHide 261)

Optimization Options

In some cases or for specific applications, you can sacrifice some complexity in the protection ir order to gain extra speed or a smaller size of your protected binary. The following options help on that:

- **Favor speed over protection**: This option reduces the complexity of the protection boot loader to increase the boot time in the protected application

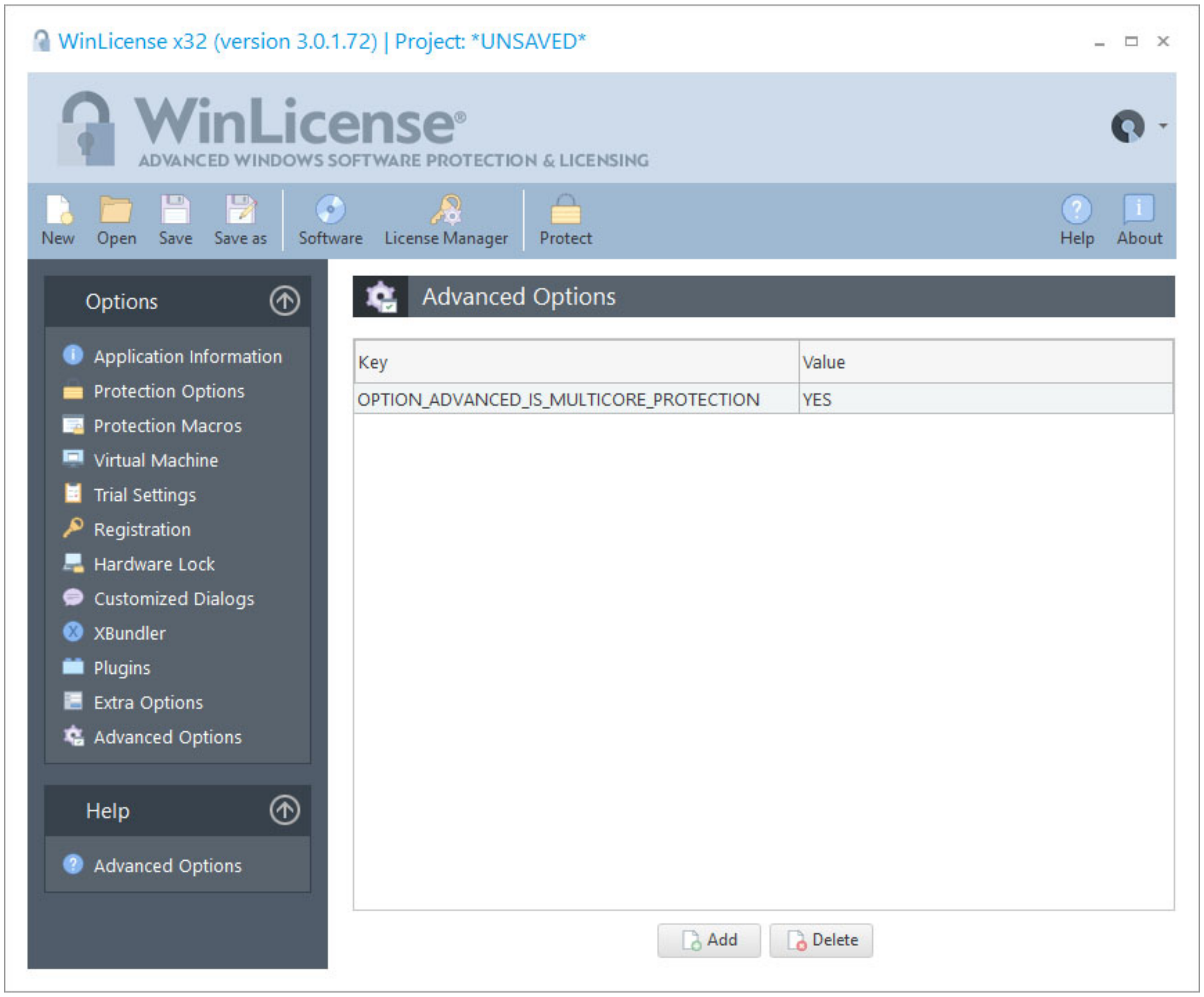
- **Favor size over protection**: This option reduces in minor degree the protection and uses a different compression algorithm in order to achieve a better compression

- **Optimize for Windows on ARM**: The Windows on ARM emulation produces some penalties when emulating specific code or data accesses. This option produces a protection code that is emulated faster on Windows on ARM with just a small reduction in complexity in the protection code.

1.3.12 Advanced Options

SecureEngine contains lots of internal options mostly related to compatibility issues with specific applications. These options are not public available as they will confuse developers if we display all current available options, which are not related to security and hardly used by most applications. Developers will feel confuse (and unsecure) if they see lots of options and he is not using them in their applications.

When there is a special compatibility issue in a specific application when applying our protection, we send the required special option to the customer so he can use it for that application. In order to add a new advanced option, just **right-click** on the Advanced Options panel and select **Add**.



Key	Value
OPTION_ADVANCED_IS_MULTICORE_PROTECTION	YES

Here we have a small list of advanced options that you can use if you consider:

- **OPTION_ADVANCED_IS_MULTICORE_PROTECTION**: You can set this option to **NO** in case that you want to use a single core to protect your application. By default multicore protection is enabled

- **OPTION_ADVANCED_SECTION_NAME:** You can set the section name (PE header) that will be appended to your protected application

- **OPTION_ADVANCED_USB_RUNTIME_CHECK_SECONDS:** You can change the number of seconds that a USB drive is checked in case that you have locked your license to a USD drive ID and you have selected the option **Detect unplug** in the Hardware Lock panel 40. The internal default value is 30

- **OPTION_ADVANCED_NETWORK_INSTANCES_DEFAULT_PORT:** (for Network Instances) You can set the default port when working with Network Instances 101 (floating licenses). The default value is 5000

- **OPTION_ADVANCED_NETWORK_INSTANCES_LIVE_TIMEOUT:** (for Network Instances) This value should be increased for heavy loaded networks (or very long distant communication) to increase the internal timeout to give some extra time for packets to arrive. The default value is 4000 (4 seconds)

- **OPTION_ADVANCED_NETWORK_INSTANCES_COUNT_BY_COMPUTERS:** (for Network Instances) If set to YES, this option will count one instance per computer, no matter if the user has launched several instances on the same computer.

- **OPTION_ADVANCED_NETWORK_INSTANCES_NO_SERVER_INSTANCE:** (for Network Instances) If set to YES, this option will not create an instance in the server. This means that the application on the server will stay on memory just waiting for clients to be connected, but your application code won't be executed.

- **OPTION_TRIAL_IS_DEBUG_MODE:** You can set this option to **YES** and the trial counters and status will be stored under the Registry key: *HKEY_CURRENT_USER/SOFTWARE/WinLicense/WLdebugTrial/Instance*. If you are doing tests on your development machine, it's better to use this option, so you can easily delete all the stored information in the Windows Registry, so the trial will be reset. When you are going to release your application, you should remove this option or set it to **NO**.

- **OPTION_ADVANCED_KEEP_DEBUG_INFO:** You can set this option to YES to keep the debug information in your protected application. This option is recommended if you want to debug crash dumps (.dmp)

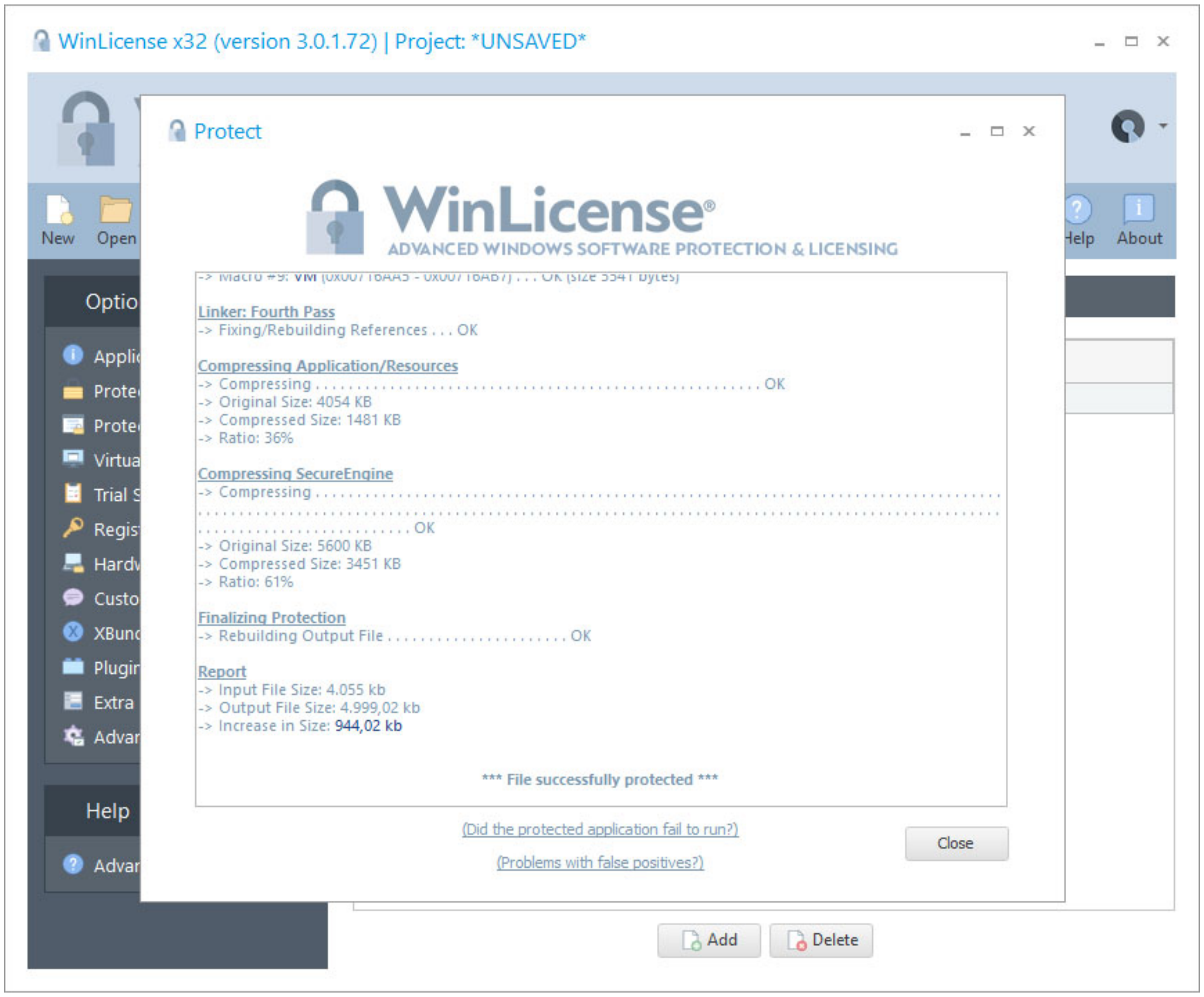
- **OPTION_ADVANCED_LAUNCH_APP_WHEN_TRIAL_EXPIRED:** When the trial is expired/manipulated, you can launch an external application before your application starts. You can also set a link to launch the default browser. Example:
  - Launch an application:

    ```
        OPTION_ADVANCED_LAUNCH_APP_WHEN_TRIAL_EXPIRED=%
    APP_FOLDER%\MyExternalApplicaiton.exe
    ```
  - Launch a web linkk:

    ```
        OPTION_ADVANCED_LAUNCH_APP_WHEN_TRIAL_EXPIRED=start
    www.yourweb.com
    ```

1.3.13 Protect Now

When you are done setting your protection options all you need to do is press the **Protect** button in the main toolbar to start the protection phase. A new window will appear to show you the current progress of the protection phase.



WinLicense x32 (version 3.0.1.72) | Project: *UNSAVED*
New
Open
Help
About
Optio
Appli
Prote
Prote
Virtua
Trial S
Regis
Hardv
Custo
XBund
Plugin
Extra
Advar
Help
Advar
Protect
WinLicense
ADVANCED WINDOWS SOFTWARE PROTECTION & LICENSING
(Did the protected application fail to run?)
(Problems with false positives?)
Close
Add
Delete

```
-> Macro #9: VM (0x00716AA5 - 0x00716AB7) . . . OK (size 5541 bytes)

Linker: Fourth Pass
-> Fixing/Rebuilding References . . . OK

Compressing Application/Resources
-> Compressing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . OK
-> Original Size: 4054 KB
-> Compressed Size: 1481 KB
-> Ratio: 36%

Compressing SecureEngine
-> Compressing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . OK
-> Original Size: 5600 KB
-> Compressed Size: 3451 KB
-> Ratio: 61%

Finalizing Protection
-> Rebuilding Output File . . . . . . . . . . . . . . . . . . . . . . OK

Report
-> Input File Size: 4.055 kb
-> Output File Size: 4.999,02 kb
-> Increase in Size: 944,02 kb

                    *** File successfully protected ***
```

1.3.14 Protecting through the command line

WinLicense can be used to protect your files through the command line in order to include the protection of your application on all of your build systems.

First you need to create a WinLicense project from the user interface. To create this project file, you need to start the WinLicense user interface and set up the protection options that you want to include in your application. After that you can invoke the following command in the command line to protect your application:

```
WinLicense /protect YourProject
```

One of the following codes will be returned:

0 Protection was successful.

1 Project file does not exist or invalid.

2 File to protect cannot be opened.

3 File already protected.

4 Error in inserted SecureEngine macros.

5 Fatal error while protecting file.

6 Cannot write protected file to disk.

7 Error while opening or reading the inserted Splash file.

8 Taggant certificate cannot be applied.

Load a project file from the command line

WinLicense also allows you to load a project into the user interface through the command line. To do this you have to invoke WinLicense in the following way:

```
WinLicense YourProject
```

After this, WinLicense user interface will appear with all the information contained in your project file and is ready to protect your applications.

Protecting a different application from the one in a project file

You can specify a different input and output application from the one that is stored in your project file when protecting via command line. Example:

```
 WinLicense /protect YourProjectFile /inputfile YourInputApplication.exe /out-
putfile YourProtectedApplication.exe
```

Protecting a different software from the one in a project file

You can specify a different software from the one that is stored in your project file when protecting via command line. Example:

```
 WinLicense /protect YourProjectFile /software YourSoftware
```

Protect several projects in parallel

If you want to protect several projects in parallel, you have to pass the "/isolate" command line parameter. The /isolate parameter will serialize accesses to the WinLicense MySQL database in order to read the specific project settings for each project that is going to be protected in parallel. Example:

```
WinLicense /protect YourProjectFile1 /isolate
WinLicense /protect YourProjectFile2 /isolate
```

Redirecting output to a file

To redirect the console output to a file, you have to use an extra parameter to avoid that WinLicense attaches itself to the current console and after that, you can use the common output redirection. The parameter to use is **/shareconsole**. This is also required when you are calling WinLicense from within Visual Studio and you want to display the information in the Output Window in Visual Studio. Example:

```
WinLicense.exe /protect YourProjectFile /shareconsole > output.txt
```

Protecting an application with a text project file

When you protect from the command line, WinLicense reads the internal database to retrieve the project information, software to protect, etc. That is, WinLicense depends on it's database (MySQL) in order to perform protection. If you don't want to rely on the WinLicense database in protection time, you can export your WinLicense project file as a text (INI) file and use that text project file to perform protection.

To generate a text project file, you can go to the Project Manager in WinLicense (click on Open Project) and select the desired project and click on **Export** button.

To protect your application from a text project file, you just need to invoke WinLicense as follows (suppose that your text project file is called *my_project.wl*)

```
WinLicense /protect my_project.wl
```

You can also specify the above extra parameters (/inputfile and/or /outputfile) if you want to use different input/output file from the one in your project file.

Opening the License Manager without the WinLicense GUI

If you just want work with the License Manager and don't want to display the main WinLicense user interface, you can use the /licensemanager parameter when launching WinLicense. Example:

```
WinLicense /licensemanager
```

Example of command line processing in a BAT file

The following example shows a BAT file that can be included in your build system to protect your applications through the command line:

```
@echo off

start /w WinLicense /protect YourProject

if errorlevel 3 goto 3
if errorlevel 2 goto 2
if errorlevel 1 goto 1
if errorlevel 0 goto 0
goto done

:0
echo Application protected successfully
goto done

:1
echo ERROR: File already protected
goto done

:2
echo ERROR: File to protect cannot be opened
goto done

:3
echo ERROR: An internal error occurred while protecting

:done
```

1.4 SecureEngine® Macros

The SecureEngine® Macros allow you to interact with your application using SecureEngine®, making your application and SecureEngine® run as a single unit.

To include SecureEngine® Macros into your application, you need to specify these macros in your application source code. When SecureEngine® is going to protect your application, it will find these macros inside your application and apply the required action to each specific macro.

The different macros that SecureEngine® offers to software developers are the following:

- Using macros in your programming language 74
- VM macro 78

- Mutate macro 79
- StrEncrypt macro 80
- Registered macro 82
- Unregistered macro 82
- Unprotected macro 83
- CheckProtection macro 84
- CheckCodeIntegrity macro 86
- CheckRegistration macro 89
- CheckVirtualPC macro 90
- CheckDebugger macro 92
- Which Macros should I use? 93

1.4.1 Using macros in your programming language

The current version of SecureEngine supports macros for native applications (developed with C/C++, Delphi, Visual Basic, etc.). Please note that these macros are not available for .NET languages or Visual Basic compiled in PCode mode.

To apply a macro to a specific block of code, you have to mark the beginning of the block with the "*MacroName_***START**" marker and the end of the block using the "*MacroName_***END**" marker.

Restrictions

A few conditions need to be satisfied in order to successfully insert SecureEngine® macros into your application. If any of these conditions are not fulfilled, WinLicense will show an error message when opening the file to be protected. The conditions are the following:

- Macros cannot be nested, that is, a macro cannot be inserted inside another macro. The following is an example of nesting macros:

```
void MyFunction(void)
{
    VM_START

      // your code

      VM_START     <--- nested!!!
```

```
            // your code

        VM_END

        // your code

    VM_END
}
```

- Each macro needs to have each corresponding "*MacroName*_**END**" delimiter.
- The code inside the macro must be at least 5 bytes in size.

Usage for specific programming languages

Specific information for Delphi developers

For Delphi, SecureEngine® macros can be inserted as a "assembly language" include file or linking with the SecureEngineSDK.pas file.

- **Macros as "include files":** In this case, macro markers appears as an external include file that will inserted with a parameter directive, *{$I filename}*. The included file will insert a special sequence of assembly code right after the *{$I filename}* directive. That sequence of assembly code will be detected and replaced by SecureEngine® in the protection phase. Notice that for 64-bit applications, Embarcadero RAD Studio for Delphi does not allow inline assembly to be inserted directly into your application. In this case, you have to insert the protection macros via function names.

- **Macros as "function names**": You can use the defined macros function names in the SecureEngineSDK.pas file in order to insert the different macro markers in your application. With this approach you are linking with the SecureEngineSDK.dll **only** in unprotected state, that is, your unprotected application will require the SecureEngineSDK.dll in order to run. Once that your application is protected, the linking with the SecureEngineSDK.dll is removed, so your protected application does not require the SecureEngineSDK.dll to run.

In the following we present a real example of how to use SecureEngine® macros in your Delphi application.

```
function TfmMain.GetCRC32(FileName: string): string;

begin

  {$I VM_Start.inc}             // the following block of code is protected with
an "VM" macro
```

```
    BuildCRCTable;
    CRC := $FFFFFFFF;

    AssignFile(F, FileName);
    FileMode := 0;
    Reset(F);

    {$I VM_End.inc}               // end of "VM" macro

    GetMem(Buffer, SizeOf(B));

    {$I Registered_Start.inc}        // the following block of code is protected
  with a "Registered" macro

    repeat
      FillChar(b, SizeOf(b), 0);
      BlockRead(F, b, SizeOf(b), e);
      for i := 0 to (e-1) do
       CRC := RecountCRC(b[i], CRC);
    until (e < 255) or (IOresult <> 0);

    {$I Registered_End.inc}          // end of "Registered" macro

    {$I Mutate_Start.inc}             // the following block of code is protected
  with an "Mutate" macro

    FreeMem(Buffer, SizeOf(B));
    CloseFile(F);
    CRC := Not CRC;
    Result := '$' + HextL(CRC);

    {$I Mutate_End.inc}               // end of "Mutate" macro

  end;
```

Specific information for C/C++ developers

For the C/C++ language you have to include the "*WinLicenseSDK.h*" in your application source code, so you can put the different macro markers inside your source code. By default, the WinLicenseSDK.h file emits inline assembly for your C/C++ 32-bit applications and function names (from within the SecureEngineSDK.dll) for your C/C++ 64-bit applications. This means that your 64-bit applications require the SecureEngineSDK.dll when you run your application **unprotected**. Once that you protect your application, the linking with the SecureEngineSDK.dll is removed, so your protected application does not require that DLL.

Following will demonstrate a real example of how to use SecureEngine® macros in your C/C++ application.

```
  LRESULT CALLBACK MainHandler(HWND hDlg, UINT message, WPARAM wParam, LPARAM
  lParam)
  {
      switch (message)
      {
      case WM_INITDIALOG:
          VM_START                        // the following block of code is protec-
  ted with a "CodeReplace" macro

          if (WLRegGetStatus(NULL) == 1)
          {
              WLRegGetLicenseInfo(Name, Company, ExtraData);
              SetDlgItemText(hDlg, IDC_NAMEEDIT, Name);
              SetDlgItemText(hDlg, IDC_COMPANYNAME, Company);
              SetDlgItemText(hDlg, IDC_EXTRAEDIT, ExtraData);
          }

          VM_END                          // end of "VM" macro

          return TRUE;

      case WM_COMMAND:

          if (LOWORD(wParam) == IDCANCEL)
          {
              MUTATE_START                        // the following block of code is
  protected with an "Encode" macro

              EndDialog(hDlg, LOWORD(wParam));

              MUTATE_END                          // end of "Mutate" macro

              return TRUE;
          }
          break;
      }
      return FALSE;
  }
```

Specific information for Visual Basic developers

For Visual Basic these macros appear as a special Visual Basic instruction that will be detected by SecureEngine® during the protection phase.

In the following we present a real example of how to use SecureEngine® macros in your Visual Basic application.

```
Private Sub CheckStatusButton_Click()

       If AppStatus <> 1 Then

                Call VarPtr("VMStart")

                TrialDaysLeftLabel.Caption = WLTrialDaysLeft
                TrialExecLeftLabel.Caption = WLTrialExecutionsLeft
                MinutesLabel.Caption = WLTrialGlobalTimeLeft
                RuntimeLabel.Caption = WLTrialRuntimeLeft

                Call VarPtr("VMEnd")

       End If

End Sub
```

1.4.2 VM macro

The **VM** macro allows you to mark regions of code that will be executed inside the SecureEngine® Virtual Machine. When the CPU is going to execute the code inside your VM macro, SecureEngine® will take control and will emulate the original code inside the macro with virtual opcodes that only the SecureEngine® Virtual machine can understand.

The VM macro is the original name used in older versions of our protection. In newer versions of our protection we recommend that you specify the name of the Virtual Machine that will be used to virtualize the code inside the START - END markers. For example, instead of using "VM_START/END" for a block of code, you should select which Virtual Machine (from the Virtual Machine panel 30) will be used to virtualize that code (example "VM_TIGER_WHITE_START/END")

We strongly recommend the use of this macro whenever possible, due to its flexibility and continuos improvement in the internal protection of these macros.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic compiled in PCode mode.

Show Delphi Macro Usage

```
{$I VM_TIGER_BLACK_START.inc}

// your code goes here

{$I VM_TIGER_BLACK_END.inc}
```

Show C/C++ Macro Usage

```
VM_TIGER_BLACK_START

// your code goes here

VM_TIGER_BLACK_END
```

Show Visual Basic Macro Usage

```
Call VarPtr("VM_START")

' your code goes here

Call VarPtr("VM_END")
```

Remarks

To make sure that you have inserted a VM macro in a right place in your application, you should be aware of the following details:

- To avoid a performance decrease, you should avoid tight loops (*FOR, WHILE, DO*...) with a big number of iterations inside the VM macro. If a specif code is called many times per second you should avoid putting a VM macro or select a lighter virtual machine macro like VM_TIGER_WHITE
- Switch/Case statements inside the macro might not work properly in some compiled applications. Notice that Switch statements are named in a different way in different programming languages ("*Case*" --> Delphi, "*Select Case*" --> VB, etc).
- Exception handling inside the macro will not work properly. You should avoid putting VM macros around *try-except* clauses. For Visual Basic, "try-except" clauses corresponds to "On Error" statements.

1.4.3 Mutate macro

The **MUTATE** macro allows you to mark regions of code that will be mutated. The original x86 machine code will be converted into equivalent but complex x86 machine code. The execution of the MUTATE macro is quite fast compared to VM macros but the level of security is quite low compared with VM macros.

The MUTATE macro is suitable for those code areas that you want to apply some obfuscation and want to keep a high performance in the execution of the mutated code.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic compiled in PCode mode.

Show Delphi Macro Usage

```
{$I Mutate_Start.inc}

// your code goes here

{$I Mutate_End.inc}
```

Show C/C++ Macro Usage

```
MUTATE_START

// your code goes here

MUTATE_END
```

Show Visual Basic Macro Usage

```
   Call VarPtr("MUTATE_START")

   ' your code goes here

   Call VarPtr("MUTATE_END")
```

Remarks

To make sure that you have inserted a MUTATE macro in a right place in your application, you should be aware of the following details:

- Switch statements inside the macro might not work properly in some compiled applications. Notice that Switch statements are named in a different way in different programming languages ("*Case*" --> Delphi, "*Select Case*" --> VB, etc).
- Exception handling inside the macro might not work properly. You should avoid putting MUTATE macros around *try-except* clauses. For Visual Basic, "try-except" clauses corresponds to "On Error" statements.

1.4.4 StrEncrypt macro

The **STR_ENCRYPT/STR_ENCRYPTW** macro allows you to mark regions of code where all referenced strings inside the region will be encrypted in protection time and decrypted in runtime when required by the target application. The decryption is performed inside the SecureEngine Virtual Machine and the location of the decrypted string is different from the original one in the unprotected application. The original location of the string is destroyed and never used again in the protected application.

If you are using Unicode strings in your application, you have to use the macro **STR_ENCRYPTW** which can process Unicode strings.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic compiled in PCode mode.

Show Delphi Macro Usage

```
   {$I StrEncrypt_Start.inc}

   // your code goes here

   {$I StrEncrypt_End.inc}
```

Show C/C++ Macro Usage

```
STR_ENCRYPT_START

// your code goes here

STR_ENCRYPT_END
```

Show Visual Basic Macro Usage

```
Call VarPtr("STR_ENCRYPT_START")

' your code goes here

Call VarPtr("STR_ENCRYPT_END")
```

Remarks

For further protection you can put a VM macro around the "STR_ENCRYPT" macro, so all your code area where your strings are used is also virtualized. Example:

```
VM_START

  STR_ENCRYPT_START

  ' your code goes here

 STR_ENCRYPT_END

VM_END
```

There are some internal options that can be added to increase the protection of the STR_ENCRYPT macro. You can add the following entries in the Advanced Options 67 panel:

- **OPTION_MACROS_ENCRYPT_STRINGS_DECRYPT_ON_HEAP=YES**

  The string will be decrypted on an allocated block in the heap

- **OPTION_MACROS_ENCRYPT_STRINGS_REENCRYPT=YES**

  The decrypted string on the heap will be destroyed when the STR_ENCRYPT_END marker is executed

1.4.5 Registered macro

The **REGISTERED** macro allows you to mark regions of code that will be executed when your application is registered. If your application is not registered, these regions of code will be kept encrypted. When your application is registered and one of these regions of code is going to be executed, SecureEngine® will decrypt that region of code with the registration information from the present license key. This macro offers a strong technique to avoid attackers from retrieving the encrypted block without having a valid license key.

NOTE:The current version of SecureEgine® does not support this function to be called for .NET languages or Visual Basic compiled in PCode mode.

Show Delphi Macro Usage

```
{$I Registered_Start.inc}

// your code goes here

{$I Registered_End.inc}
```

Show C/C++ Macro Usage

```
REGISTERED_START

// your code goes here

REGISTERED_END
```

Show Visual Basic Macro Usage

```
Call VarPtr("REGISTERED_START")

' your code goes here

Call VarPtr("REGISTERED_END")
```

1.4.6 Unregistered macro

The **UNREGISTERED** macro allows you to mark regions of code that will be executed when your application is running in trial mode. If your application is registered, these regions of code will be kept encrypted. When your application is running in trial mode and one of these regions of code is going to be executed, SecureEngine® will decrypt that region of code.

NOTE:The current version of SecureEgine® does not support this function to be called for .NET languages or Visual Basic compiled in PCode mode.

Show Delphi Macro Usage

```
{$I Unregistered_Start.inc}

// your code goes here

{$I Unregistered_End.inc}
```

Show C/C++ Macro Usage

```
UNREGISTERED_START

// your code goes here

UNREGISTERED_END
```

Show Visual Basic Macro Usage

```
Call VarPtr("UNREGISTERED_START")

' your code goes here

Call VarPtr("UNREGISTERED_END")
```

1.4.7 Unprotected macro

The **UNPROTECTED** macro allows you to mark regions of code that will be ONLY executed when your application is not yet protected. Once your application is protected, the code inside the macro will not be executed, basically the code inside the macro markers is destroyed and jumped to avoid execution. This macro is only necessary to avoid releasing unprotected applications by mistake.

NOTE:The current version of SecureEgine® does not support this function to be called for .NET languages or Visual Basic applications.

Show Delphi Macro Usage

```
{$I Unprotected_Start.inc}

// your code goes here

{$I Unprotected_End.inc}
```

Show C/C++ Macro Usage

```
UNPROTECTED_START
```

```
   // your code goes here

   UNPROTECTED_END
```

1.4.8 CheckProtection macro

The **CHECK_PROTECTION** macro allows you to check if your application has been partially unpacked or some protection engines have been attacked by a cracker. This macro offers communication between the protected application and the SecureEngine protection.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic applications.

The CHECK_PROTECTION macro can be called from inside other macros. In fact, it's highly recommend to call the CHECK_PROTECTION macro from inside VM 78 macros.

The CHECK_PROTECTION macro has a special syntax:

CHECK_PROTECTION (user_variable, user_value)

Where "*user_variable*" is any **local** or **global** variable in the application and "*user_value*" is any **immediate value (constant value)**. The way that it works is the following:

- The CHECK_PROTECTION macro is called.
- SecureEngine takes control of the processor and make special checks to know if the application has been tampered.
- If the application is not tampered, SecureEngine sets "user_variable" equal to "user_value".
- If the application is tampered, SecureEngine does not set "user_variable". You should take care of initializing "user_variable" to something else from "user_value".
- SecureEngine returns control to the protected application. The protected application should check the value of "user_variable" and execute the desired action if the application has been tampered.

If you detect that your application has been tampered, please, consider the following practices:

- Avoid taking an immediate action, like displaying a message or crashing the application. If you take an immediate action, the cracker will know where the problematic code is located and will focus all his attention at that point, trying to figure out the root of the problem in that code.

- Avoid displaying messages saying that the application has been tampered. Instead, make a "late" crash (see below) or display a strange error message at a later point in your application.
- Produce a "late crash" or malfunction. That is, if you detect that your application has been tampered, you mark special variables (or similar action) in your code. At a later point in your application, you crash your application or initialize further structures in a wrong way, so, your application won't work as expected. For example, suppose that you are protecting a CD burning application. When your application is initializing, you call "CHECK_PROTECTION" macro to determine if the application is tampered or not. If it's tampered, you won't take any action yet, but instead, you will wait for the CD recording process to burn random or incorrect data into the CD.
- Use VM 78 macros in all those places where you call CHECK_PROTECTION and where you check if the application was tampered. Also, if you decide to produce a "late" crash or malfunction, that code which produces the crash or malfunction should go inside VM or CodeReplace macros.

Show C/C++ Macro Usage

```
int MyCheckVar;

VM_START

    // your code goes here

    CHECK_PROTECTION(MyCheckVar, 0x12345678)

    // your code goes here

    if (MyCheckVar != 0x12345678)
      printf("We are tampered!");

VM_END
```

Show Delphi Macro Usage

```
var
  MyCheckVar: Integer;

begin

{$I VM_Start.inc}

    // your code goes here

    {$I CheckProtection_Prolog.inc}
    asm
      push 11111111                      // 11111111 is our special constant
      pop  MyCheckVar                    // SecureEngine will set "MyCheckVar" to
    11111111 if protection is OK
```

```
      end;
      {$I CheckProtection_Epilog.inc}

      // your code goes here

      if MyCheckVar <> 11111111 then
        ShowMessage("We are tampered!");

  {$I VM_End.inc}
```

Advises about how to use this macro

- Put the CHECK_PROTECTION macro inside VM or CodeReplace macros.
- Think always that the first attack from a cracker is just directly jump over your VM / CodeReplace macro (that is, the code inside the macro is not executed), so you should make sure that inside the macro you put code that is necessary for your application to run correctly.
- You don't have to call the CHECK_PROTECTION macro periodically, just make sure that it's executed at any time in your application.
- You can put as many CHECK_PROTECTION macros as desired, but we recommend you just putting a few of them (about 5 of them) in different routines in your application.

1.4.9 CheckCodeIntegrity macro

The **CHECK_CODE_INTEGRITY** macro allows you to check if the code section of your protected application has been patched in runtime (using for example an memory patcher). This macro offers communication between the protected application and the SecureEngine protection.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic applications.

The CHECK_CODE_INTEGRITY macro can be called from inside other macros. In fact, it's highly recommend to call the CHECK_CODE_INTEGRITY macro from inside VM 78 macros.

The CHECK_CODE_INTEGRITY macro has a special syntax:

CHECK_CODE_INTEGRITY (user_variable, user_value)

Where "*user_variable*" is any **local** or **global** variable in the application and "*user_value*" is any **immediate value (constant value)**. The way that it works is the following:

- The CHECK_CODE_INTEGRITY macro is called.

- SecureEngine takes control of the processor and make special checks to know if the code section of your application has been patched.
- If the code section of the application is not patched, SecureEngine sets "user_variable" equal to "user_value".
- If the application is patched, SecureEngine does not set "user_variable". You should take care of initializing "user_variable" to something else from "user_value".
- SecureEngine returns control to the protected application. The protected application should check the value of "user_variable" and execute the desired action if the code section of the application has been patched.

If you detect that the code section of your application has been tampered, please, consider the following practices:

- Avoid taking an immediate action, like displaying a message or crashing the application. If you take an immediate action, the cracker will know where the problematic code is located and will focus all his attention at that point, trying to figure out the root of the problem in that code.
- Avoid displaying messages saying that the application has been tampered. Instead, make a "late" crash (see below) or display a strange error message at a later point in your application.
- Produce a "late crash" or malfunction. That is, if you detect that your application has been tampered, you mark special variables (or similar action) in your code. At a later point in your application, you crash your application or initialize further structures in a wrong way, so, your application won't work as expected. For example, suppose that you are protecting a CD burning application. When your application is initializing, you call "CHECK_CODE_INTEGRITY" macro to determine if the application code is patched or not. If it's patched, you won't take any action yet, but instead, you will wait for the CD recording process to burn random or incorrect data into the CD.
- Use VM 78 or macros in all those places where you call CHECK_CODE_INTEGRITY and where you check if the application code was patched. Also, if you decide to produce a "late" crash or malfunction, that code which produces the crash or malfunction should go inside VM or CodeReplace macros.

Show C/C++ Macro Usage

```
int MyCheckVar;

VM_START
```

```
      // your code goes here

      CHECK_CODE_INTEGRITY(MyCheckVar, 0x12345678)

      // your code goes here

      if (MyCheckVar != 0x12345678)
        printf("Application code is patched!");

  VM_END
```

Show Delphi Macro Usage

```
  var
    MyCheckVar: Integer;

  begin

  {$I VM_Start.inc}

      // your code goes here

      {$I CheckCodeIntegrity_Prolog.inc}
      asm
        push 11111111                         // 11111111 is our special constant
        pop  MyCheckVar                       // SecureEngine will set "MyCheckVar" to
      11111111 if protection is OK
      end;
      {$I CheckCodeIntegrity_Epilog.inc}

      // your code goes here

      if MyCheckVar <> 11111111 then
        ShowMessage("We are tampered!");

  {$I VM_End.inc}
```

Advises about how to use this macro

- Put the CHECK_CODE_INTEGRITY macro inside VM macros.
- You should call the CHECK_CODE_INTEGRITY at specific points in your application code. You could also call it from a thread which calls that macro periodically (once each 30-60 seconds).
- For applications with a big code section, this macro might take some time to be executed. If you want to increase the speed when calling this macro, you can insert the following option in the Advanced Options 67 panel:

```
OPTION_MACROS_FAST_CHECK_CODE_INTEGRITY=YES
```

1.4.10 CheckRegistration macro

The **CHECK_REGISTRATION** macro allows you to check if your application has been registered with a valid license key. You can also know if your application is registered via WLRegGetStatus 161. The CHECK_REGISTRATION macro can be used as a double check to make sure that your application has not been tampered.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic applications.

The CHECK_REGISTRATION macro can be called from inside other macros. In fact, it's highly recommend to call the CHECK_REGISTRATION macro from inside VM 78 macros.

The CHECK_REGISTRATION macro has a special syntax:

CHECK_REGISTRATION (user_variable, user_value)

Where "*user_variable*" is any **local** or **global** variable in the application and "*user_value*" is any **immediate value (constant value)**. The way that it works is the following:

- The CHECK_REGISTRATION macro is called.
- SecureEngine takes control of the processor and make special checks to know if your application is correctly registered.
- If your application is correctly registered, SecureEngine sets "user_variable" equal to "user_value".
- If the application is not registered, SecureEngine does not set "user_variable". You should take care of initializing "user_variable" to something else from "user_value".
- SecureEngine returns control to the protected application. The protected application should check the value of "user_variable" and execute the desired action if the application is not registered.

Show C/C++ Macro Usage

```
int MyCheckVar;

VM_START

   // your code goes here

   CHECK_REGISTRATION(MyCheckVar, 0x12345678)

   // your code goes here
```

```
    if (MyCheckVar != 0x12345678)
      printf("Application is not registered");

VM_END
```

Show Delphi Macro Usage

```
var
  MyCheckVar: Integer;

begin

{$I VM_Start.inc}

    // your code goes here

    {$I CheckRegistration_Prolog.inc}
    asm
      push 11111111                     // 11111111 is our special constant
      pop  MyCheckVar                   // SecureEngine will set "MyCheckVar" to
    11111111 if application is registered
    end;
    {$I CheckRegistration_Epilog.inc}

    // your code goes here

    if MyCheckVar <> 11111111 then
      ShowMessage("Application is not registered!");

{$I VM_End.inc}
```

1.4.11 CheckVirtualPC macro

The **CHECK_VIRTUAL_PC** macro allows you to check if your application is running under VMWare/VirtualPC.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic applications.

The CHECK_VIRTUAL_PC macro can be called from inside other macros.

The CHECK_VIRTUAL_PC macro has a special syntax:

CHECK_VIRTUAL_PC (user_variable, user_value)

Where "*user_variable*" is any **local** or **global** variable in the application and "*user_value*" is any **immediate value (constant value)**. The way that it works is the following:

- The CHECK_VIRTUAL_PC macro is called.
- SecureEngine takes control of the processor and make special checks to know if your application is running under VMWare/VirtualPC.

- If your application is **not** running under VMWare/VirtualPC, SecureEngine sets "user-_variable" equal to "user_value".
- If the application is running under VMWare/VirtualPC, SecureEngine does not set "user_variable". You should take care of initializing "user_variable" to something else from "user_value".
- SecureEngine returns control to the protected application. The protected application should check the value of "user_variable" and execute the desired action if the application is running under VMWare/VirtualPC.

Show C/C++ Macro Usage

```
int MyCheckVar;

VM_START

    // your code goes here

    CHECK_VIRTUAL_PC(MyCheckVar, 0x12345678)

    // your code goes here

    if (MyCheckVar != 0x12345678)
      printf("Application is running under VMWare/VirtualPC");

VM_END
```

Show Delphi Macro Usage

```
var
  MyCheckVar: Integer;

begin

{$I VM_Start.inc}

    // your code goes here

    {$I CheckVirtualPC_Prolog.inc}
    asm
      push 11111111                        // 11111111 is our special constant
      pop  MyCheckVar                      // SecureEngine will set "MyCheckVar" to
    11111111 if VMWare not present
    end;
    {$I CheckVirtualPC_Epilog.inc}

    // your code goes here

    if MyCheckVar <> 11111111 then
      ShowMessage("Application is running under VMWare/VirtualPC!");

{$I VM_End.inc}
```

1.4.12 CheckDebugger macro

The **CHECK_DEBUGGER** macro checks if your application is running under a debugger.

NOTE: The current version of SecureEgine® does not support this macro for .NET languages or Visual Basic applications.

The CHECK_DEBUGGER macro can be called from inside other macros.

The CHECK_DEBUGGER macro has a special syntax:

CHECK_DEBUGGER (user_variable, user_value)

Where "*user_variable*" is any **local** or **global** variable in the application and "*user_value*" is any **immediate value (constant value)**. The way that it works is the following:

- The CHECK_DEBUGGER macro is called.
- SecureEngine takes control of the processor and make special checks to know if your application is running under a debugger.
- If your application is **not** running under a debugger, SecureEngine sets "user_variable" equal to "user_value".
- If the application is running under a debugger, SecureEngine does not set "user_variable". You should take care of initializing "user_variable" to something else from "user_value".
- SecureEngine returns control to the protected application. The protected application should check the value of "user_variable" and execute the desired action if the application is running under a debugger.

Show C/C++ Macro Usage

```
int MyCheckVar;

VM_START

   // your code goes here

   CHECK_DEBUGGER(MyCheckVar, 0x12345678)

   // your code goes here

   if (MyCheckVar != 0x12345678)
     printf("Application is running under a debugger!");

VM_END
```

Show Delphi Macro Usage

```
var
  MyCheckVar: Integer;

begin

{$I VM_Start.inc}

    // your code goes here

    {$I CheckDebugger_Prolog.inc}
    asm
      push 11111111                        // 11111111 is our special constant
      pop  MyCheckVar                      // SecureEngine will set "MyCheckVar" to
    11111111 if VMWare not present
    end;
    {$I CheckDebugger_Epilog.inc}

    // your code goes here

    if MyCheckVar <> 11111111 then
      ShowMessage("Application is running under a debugger!");

{$I VM_End.inc}
```

1.4.13 Which Macros should I use?

It is normal that a programmer feel lost when deciding which macros he/she should use. We recommend mostly using our virtualization macros (VM, TIGER_VM, FISH_VM, etc) as they offer the biggest protection in latest versions of our protection.

It's not a good idea to insert multiple (different) protection virtual machines as the final size of your application will growth noticeably. A good approach is to use a lighter VM (like FISH/TIGER) for your code that needs to be quite protected but also executed fast and use a heavier VM (like FISH_BLACK, PUMA, SHARK, ...) for your code that needs to be highly protected and you can afford that extra time that it takes to execute the virtualized code under the heavier VM.

When you call the WinLicense SDK functions, it's a good idea to put a virtual machine macro (VM, TIGER, FISH...) around the code that calls the WinLicense SDK function.

1.5 Licensing with WinLicense

WinLicense provides a flexible licensing system to satisfy most developers' needs. The following sections describe the different registration ways to register a protected application.

- Overview 94
- File keys 94

- Registry keys 95
- Text keys 95
- SmartActivate keys 96
- Generating licenses 97
- Setting license restrictions
- Banning Licenses 97
- Customizing features in licenses 100

1.5.1 Overview

WinLicense offers several ways to register an application, giving total freedom to developers to decide which registration scheme prefer to register their applications. At first glance, it could be a bit confusing for new WinLicense user how to use WinLicense to register their protected applications.

Basically, an application can be register with a **File** key or a **Registry** key (the license names can be customized in the Registration panel 36). The protection strength is totally the same in either File or Registry keys, it's up to developer to select the type of key that he prefers for his application (or select both types if desired)

WinLicense also offers two other methods to license an application (**Text** Keys and **SmartActivate** keys) but at the end, Text and SmartActivate keys need to be converted into a File or Registry key programmatically (using the WinLicense SDK 105) to fully register an application.

1.5.2 File keys

File keys is one of the basic ways to register an application with WinLicense. If you decide to register your application with a File key, your customers have to place the file key in the same folder as your protected application (you can also change the folder where your application expects the File key).

In the Registration 36 panel, you can select the name of the expected File key (**Single File** edit box). Notice that you can also use special folder constants to specify a different folder to search for your File keys. Please, refer to the following entry 332 for information about different locations where you can put a license.

Once that a File key is present in the same folder as your protected application (or a different folder that you select), WinLicense will check the File key when your application boots up and will try to register your application if the File key is correct and it's not expired (in case that expiration options were introduced in the File key).

1.5.3 Registry keys

Registry keys is one of the basic ways to register an application with WinLicense. If you decide to register your application with a Registry key, you have to ship a **.reg** file which will contain the Registry key information. Your customers can double click in the .reg file and the Registry key information will be inserted into the Windows Registry automatically.

In the Registration 36 panel, you can select the Registry key name and Value name where the Registry key information will be stored in the Windows Registry. Notice that you can select to install the license in either *HKEY_LOCAL_MACHINE* or *HKEY_CURRENT_USER*. If you select HKEY_LOCAL_MACHINE, the license will be inserted for all users in a specific computer, but the user must have administrator's rights to install the license into the Windows Registry. If you select HKEY_CURRENT_USER, the license will be inserted for just the current user but it will not require administrator's rights to install the license.

Once that a Registry key is fully installed into the Windows Registry, WinLicense will check the Registry key when your application boots up and will try to register your application if the Registry key is correct and it's not expired (in case that expiration options were introduced in the Registry key).

1.5.4 Text keys

Text keys is a more complex way to license an application, but it gives much transparency for the final user, who installs the license.

The problem with File or Registry keys is that for File keys, the final user needs to explicitly place the File key into the same folder as your protected application and for Registry keys, the user needs to double-click on a *.reg* file to insert the license information into the Registry.

To solve the above problems, WinLicense introduces Text keys. Text keys are given in ASCII format and give the developer the chance to create a Registration form in his application where the Text key can be introduced. The final user will insert the Text key in the Registration form and the developer will call the WinLicense SDK function WLRegNormalKeyCheck 169 to check if the introduced Text key is correct.

Once that the Text key has been checked and it's correct, the developer can either insert the Text key as File key (calling WLRegNormalKeyInstallToFile 172) or as Registry key (calling WLRegNormalKeyInstallToRegistry 174). Notice that if you decide to install a Text key as File key, you have to enable the option "**Single file**" in the Registration 36 panel and if you decide to install the Text key as Registry key, you have to enable the option "**Registry**" in the Registration 36 panel.

Basically, what we are doing with Text keys is to avoid the final user to place a File key or a Registry key into a specific place, so you do all those steps from inside your application, giving the final user more transparency to register your applications.

1.5.5 SmartActivate keys

SmartActivate keys have the same transparency for the final user as Text keys 95 but they produce a small registration code, which is suitable for many developers when they want to send their licenses via SMS, FAX, etc, or they just prefer a short code registration serial.

SmartActivate keys are given as a serial number in ASCII format. The SmartActivate key can be inserted into your Registration form by your customers when they are going to register your application. Once that the SmartActivate key is inserted, the developer will call the WinLicense SDK function WLRegSmartKeyCheck 178 to check if the introduced SmartActivate key is correct.

Once that the SmartActivate key has been checked and it's correct, the developer can either insert the SmartActivate key as File key (calling WLRegSmartKeyInstallToFile 182) or as Registry key (calling WLRegSmartKeyInstallToRegistry 189). Notice that if you decide to install a SmartActivate key as File key, you have to enable the option "**Single file**" in the Registration 36 panel and if you decide to install the SmartActivate key as Registry key, you have to enable the option "**Registry**" in the Registration 36 panel.

WinLicense offers two types of SmartActivate keys:

- **Static SmartActivate keys**: They are SmartActivate keys with a fix and short length. These keys cannot include all possible license restrictions offered by WinLicense (like runtime expiration, country locking, Network instances, etc) and they are not as strong as Dynamic SmartActivate keys.

  This is an example of a Static SmartActivate key: *77172C78-D80A4A04-1CD70B1F-493E5EC2-9DA63776-D530B309-07E0*

- **Dynamic SmartActivate keys**: They offer a stronger security layer (based in elliptic curves and other crypto algorithms) and can include all type of license restrictions offered by WinLicense. They size of these keys are larger than Static SmartActivate keys and they are also length-variable (depending on restrictions inserted in the license).

  This is an example of a Dynamic SmartActivate key: *NF5M5RNA-SIYKK5VX-KQ7MM74D-GMU7FHFP-FUYC4AQV-AC7JLQDC-PWTYRM25-VEFT374O-XUV2PZGB-OQBBKAFH-OIIOYQBW-NR5L624P-Q2AZN7CC-HSCGEWIQ*

Dynamic SmartActivate keys were introduced in WinLicense 2.0 and we recommend you to select them instead of Static SmartActivate keys, to get more security and flexibility in your SmartActivate keys.

1.5.6 Generating licenses

WinLicense offers several ways to generate licenses for your protected applications. The current version of WinLicense supports the following ways:

- Via the WinLicense **License Manager**: You can create licenses for your applications from the License Manager panel in WinLicense.
- Via **WinLicense SDK** functions: You can use the WinLicenseSDK 194 APIs in WinLicenseSDK.dll to generate licenses for your application.
- Via **Custom WinLicense SDK** functions: When you protect your application, WinLicense will generate a specific generator DLL 69 to help you generating licenses for your application.
- Via ***Exported Generator*** application: When you protect your application, WinLicense will create a small application 69 (with database support) to help you manage your licenses and customers.
- Via our **C ANSI source code** generator: Our customers can get access to our C ANSI source code generator to generate licenses under any platform (Windows, UNIX, Linux, Mac, etc).

1.5.7 Banning Licenses

Sometimes it's necessary to set a license as stolen, leaked, etc. so you can disable that license from being used in your current or new versions of your application.

Disabling licenses before protection

With WinLicense it's easy to disable a license for a new protected version of your application. Basically, you set the licenses to disable before the protection stage, in your project settings. The protected application won't accept those licenses and will generate the event MSG_ID_LICENSE_STOLEN (Customized Dialogs 43 panel) in runtime.

Before protecting your application, you can disable specific licenses from the Advanced Options 67 panel in WinLicense:

For common File/Registry licenses:

- Insert an entry in the Advanced Options panel with the number of licenses that you want to ban. Example:

```
OPTION_STOLEN_LICENSES_NUMBER_NAMED_LICENSES=2
```

- For each license that you want to ban, insert 3 entries in the Advanced Options panel with the User/Company/Custom Data of the license to ban. Example:

```
OPTION_STOLEN_LICENSES_USER_NAME_AT_0=Alexander
OPTION_STOLEN_LICENSES_COMPANY_NAME_AT_0=ax1@alex.co.com
OPTION_STOLEN_LICENSES_CUSTOM_DATA_AT_0=Alex's custom data

OPTION_STOLEN_LICENSES_USER_NAME_AT_1=Manuel Sanchez
OPTION_STOLEN_LICENSES_COMPANY_NAME_AT_1=msanchez@m-
sanch.co.com
OPTION_STOLEN_LICENSES_CUSTOM_DATA_AT_1=Another custom Data\nNew
Line
```

For SmartKeys:

- Insert an entry in the Advanced Options panel with the number of licenses that you want to ban. Example:

```
OPTION_STOLEN_LICENSES_NUMBER_SMARTKEY_LICENSES=2
```

- For each license that you want to ban, insert 4 entries in the Advanced Options panel with the User/Company/Custom Data of the license to ban. Example:

```
OPTION_STOLEN_LICENSES_USER_NAME_AT_0=Alexander
OPTION_STOLEN_LICENSES_COMPANY_NAME_AT_0=ax1@alex.co.com
OPTION_STOLEN_LICENSES_CUSTOM_DATA_AT_0=Alex's custom data
OPTION_STOLEN_LICENSES_SMARTKEY_AT_0=AABB-1821-A45B-81C4-........

OPTION_STOLEN_LICENSES_USER_NAME_AT_1=Manuel Sanchez
OPTION_STOLEN_LICENSES_COMPANY_NAME_AT_1=msanchez@m-
sanch.co.com
OPTION_STOLEN_LICENSES_CUSTOM_DATA_AT_1=Another custom Data\nNew
Line
OPTION_STOLEN_LICENSES_SMARTKEY_AT_1=1829-8D1A-45C3........
```

In case that you are using the Custom Data with several lines, you have to can put "\n" to specify a new line.

The above process can be quite time consuming if you are going to insert multiple licenses to be set as banned/stolen. You can create an external configuration file with the above entries, so you can auto generate the external configuration file with a simple application. If you are going to use an external configuration file, you have to insert an entry in the Advanced Options panel to point to the location of your configuration file. Example:

```
OPTION_ADVANCED_EXTERNAL_PROJECT_OPTIONS=Path\YourConfigurationFile.txt
```

Notice that you can use the special path constants 363 to work with relative paths.

There is a special advanced option in case that you want to ignore the custom data field from the license that you want to ban. For example, you deliver several licenses to a specific customer with different Custom Data information. Now you decide to ban that customer but you don't want to create multiple entries for each specific Custom Data that you generated for that specific customer, just create a single entry for that customer. To achieve that, just insert the following entry in the Advanced Options panel:

```
OPTION_STOLEN_LICENSES_IGNORE_CUSTOM_DATA=YES
```

Now you just set the license user name and company name to ban, without inserting an entry for the custom data information. Example:

```
OPTION_STOLEN_LICENSES_USER_NAME_AT_0=Alexander
OPTION_STOLEN_LICENSES_COMPANY_NAME_AT_0=ax1@alex.co.com

OPTION_STOLEN_LICENSES_USER_NAME_AT_1=Manuel Sanchez
OPTION_STOLEN_LICENSES_COMPANY_NAME_AT_1=msanchez@msanch.co.com
```

NOTE: In case that you use an external configuration file, you have to save it as Unicode (UTF-16). For example, in Windows Notepad save with encoding "UNICODE".

Disabling a license in runtime

You can also disable a license in runtime in the current protected application. In order to achieve this, you have to call the WinLicense SDK from your protected application. The function WLRegDisableCurrentKey 147 will disable the current license from being used in your current application (and also in your new protected versions of your application where you keep the same "License Hash 363")

There are some cases, where the function WLRegDisableCurrentKey 147 might be too strict for your specific scenario. Suppose that you want to ban a license for a specific customer but only for the current version of your application, but you want that the license can still be used in older or newer versions of your application. In that case, you should call instead the function WLRegDisableKeyCurrentInstance 148 to achieve that goal.

1.5.8 Customizing features in licenses

A common question by developers when using WinLicense is how they can set up specific options for each license that they generate. For example, a developer sells different types of licenses to allow some features to be present or not depending on the type of license. He wants to create licenses to put the application in Basic Mode, Advance Mode and Enterprise Mode. Most licensing systems on the market allow doing that by setting special bits when creating the license, so the developer has to interpret which bits are active in the current license to know which type of license is present.

WinLicense offers a more flexible approach and allows you to insert a buffer (ASCII or UNICODE) of up to 6400 bytes when generating the license (in the **Custom Data** field). You can put any information that you want and you can check it in runtime by calling the function WLRegGetLicenseInfo 156. There are not restrictions about the format of the Custom Data information that you insert. The information that you set in the Custom Data when creating the license, it's the one that it's returned by the WLRegGetLicenseInfo 156 function.

1.6 License Restrictions

WinLicense accepts different type of restrictions for the generated licenses. You can create licenses with no expiration restrictions at all (so the license will work forever) or add one or more different expiration restrictions to a single license, so your license can expire for example by number of days or by number of executions (the one that happens first)

- **Days Expirations**: This is the number of days that your license can run since it was initially used. When your customer runs your application with a license with days expiration, WinLicense will keep track of the day that the license was first used. After that, the license will expire after the given number of days. When the license expires by days, WinLicense will signal the event **MSG_ID_LICENSE_DAYS_EXPIRED** (from the Customized Dialogs 43 panel)

- **Date Expiration**: You can set a specific expiration date for your license. The license will expire when the expiration date arrives, even if the user has no launched the application before that date. When the license expires by date, WinLicense will signal the event **MSG_ID_LICENSE_DATE_EXPIRED** (from the Customized Dialogs 43 panel)

- **Executions**: You can restrict a license to be only used a specific number of times on a machine. Each time that your application is launched, the internal execution counter will add one execution. When there are no more executions left, WinLicense will signal the event **MSG_ID_LICENSE_EXECUTIONS_EXPIRED** (from the Customized Dialogs 43 panel)

- **Runtime**: You can restrict your license to be only used a specific amount of minutes in memory. Each time that the application is launched, the runtime expiration starts again from zero till it reaches the specific limit in the license. When your application has been running in memory during the specific runtime limit, WinLicense will signal the event **MSG_ID_LICENSE_RUNTIME_EXPIRED** (from the Customized Dialogs 43 panel)

- **Global Time**: You can set the maximum time (minutes) that a license can be used on a machine. For example, if you set 180 minutes as global time expiration, your customer can launch your application multiple times but WinLicense keeps a counter of the number of minutes that your application has been in memory. Once that the customer has been running your application during more than 180 minutes (no matter if he just launched one time and consumed the 180 minutes or it took him 3 months to reach the 180 minutes) your license will be expired. When the global time is expired, WinLicense will signal the event **MSG_ID_LICENSE_GLOBALTIME_EXPIRED** (from the Customized Dialogs 43 panel)

- **Install Before Date**: You can force a license to be installed on a machine before a specific date. If your customer installs the license after the specific date, WinLicense will signal the event **MSG_ID_LICENSE_INSTALLATION_TIME_EXCEEDED** (from the Customized Dialogs 43 panel)

- **Country Locking**: This option is used to create a restricted license that can only be used in a specific country. Notice that the country is determined by the Windows Language settings and it's not a real proof that the user is really running your application from a specific country. If the current language does not match the specific country inside the license, WinLicense will signal the event **MSG_ID_LICENSE_COUNTRY_ERROR** (from the Customized Dialogs 43 panel)

- **Network Instances**: This option is used to limit the number of instances inside a network. Please, refer to the Network Instances (Floating licenses) 101 section for more information.

1.6.1 Network Instances (Floating licenses)

You can generate licenses that are restricted to a specific number of instances over a local network, an intranet or virtual private network. The Network Instances feature works on protected EXE files and with some "restrictions" on DLLs (refer below).

Running an application with Network Instances

The Network Instances module works in Client-Server model. These are the basic steps to make the whole model work:

1. Your customer has to select one of his machines to determine which one will behave as server and send you the Hardware ID for that "server" machine. After that, you create a license with the option Network Instances checked (with the maximum number of instances) and also locking the license to the given (server) hardware ID.

2. The customer now runs the instance on the server. To do so he has to launch your protected application on the server as follows (suppose that the server IP is 192.168.1.22 and we want to work on port 5100)

   MyApplication.exe /wl_net_gui /wl_net_server_ip 192.168.1.22 /wl_net_port 5100

   - The parameter **/wl_net_gui** launches a small window where the customer can see the number of clients (instances) that are running

   - The parameter **/wl_net_server_ip** should match the server IP address. This is because the server will act as a server and as the *first client* (first instance)

   - The parameter **/wl_net_port** specifies the port where the server and clients talk. The default value is 5000. If this parameter is not used it will use the default port (5000)

   - If the user doesn't want that the server instance will act as a client instance he can launch your protected application with the /wl_net_server_no_instance. Example:

     MyApplication.exe /wl_net_gui /wl_net_server_no_instance

     Notice that in this case the server IP is not required and your application will "not start". That is, your application will stay dormant in memory just waiting for clients to be connected.

3. Other computers on the network can now launch your protected application (while the instance in the server is running). To do so, clients have to launch your application with the following parameters (Suppose that the server IP is 192.168.1.22 and listening on port 5100)

   MyApplication.exe /wl_net_server_ip 192.168.1.22 /wl_net_port 5100

There are other extra parameters related with time when the server and clients communicate:

- **/wl_net_timeout** *time_in_milliseconds*

The "wl_net_timeout" should be increased for heavy loaded networks (or very long distant communication) to increase the internal timeout to give some extra time for packets to arrive. The default value is 4000 (4 seconds)

Using a Configuration File

You can create a configuration file instead of passing arguments via the command line. To do so, please, add a special option in the Advanced Options 67 panel with the file name of your configuration file. Example:

OPTION_ADVANCED_NETWORK_FILE_SETTINGS_NAME=network_settings.ini

For the file name location, you can use special folder constants like: `%APP_FOLDER%, %WIN_FOLDER%, %WINSYS_FOLDER%, %USER_DOCS%, %LOCAL_APP_DATA%, %USER_APP_DATA%, %COMMON_APP_DATA%, %TEMP_FOLDER%, %PUBLIC_DOCS%`

For the file name you can use special constants like: `%APP_NAME%, %SOFT_NAME%`

Example:

OPTION_ADVANCED_NETWORK_FILE_SETTINGS_NAME=%USER_APP_DATA%\MySoftware\%APP_NAME%.ini

The configuration file is a text file like the following:

```
[NETWORK SETTINGS]

wl_net_server_no_instance=no
wl_net_server_ip=192.168.1.22
wl_net_port=5000
wl_net_timeout=2000
wl_net_gui=yes
```

Please, do not forget the "[NETWORK SETTINGS]" section name and all the keys and values in lowercase.

Possible Events (errors)

When a client instance is launched, WinLicense checks if a server instance is running and also count the number of clients running on the network. There are several events that can arise when working with floating licenses. All these events can be edit in the Customized Dialog panel or being handled by yourself (from a plugin DLL or from the WinLicense SDK)

- **MSG_ID_NETWORK_INSTANCES_NO_SERVER**: This event occurs when a client is trying to launch your protected application but there is no a server instance running or it cannot communicate with the server instances (firewall blocking network access, network misconfiguration, etc).

- **MSG_ID_NETWORK_INSTANCES_NO_HW_LOCKED**: As described above, when working with network instances, you have to also hardware lock your license (with the server hardware id). In case that a floating license is found and the license is not hardware locked, WinLicense will signal this event.

- **MSG_ID_NETWORK_INSTANCES_CANNOT_START_SERVER**: This event occurs when the server instance cannot be started (probably due to firewall restrictions, network disconnected, etc). Notice that when this message appears, it means that the Hardware ID in the license is matching with the one on the server, the issue is probably coming from an incorrect IP address that has been passed as parameter. Please, make sure that you are passing the current IP address of the local machine (that is going to act as a server)

- **MSG_ID_NETWORK_INSTANCES_NO_MORE**: This event occurs when a new client is launching your application but the maximum number of network instances has been reached.

WinLicense SDK when working with floating licenses

There are specific functions in the SDK to work with floating licenses:

- WLRegNetInstancesMax 168: Retrieves the maximum number of instances allowed in the current license.

- WLRegNetInstancesGet 167: Retrieves the current number of instances running in a network.

Detect if a user is cheating

WinLicense allows a client to run in case that the maximum number of instances has not been reached. Once that the client instance starts, a customer might want to disconnect (or block) the network access to your running application, so the server will think that the current client was closed and there is again a free entry for a new client.

At the moment, the only way to detect if the running client instance has been "unplugged" from the network is by calling the function WLRegNetInstancesGet 167. If the function returns zero, that means the there is no connection with the server, so you can determined that the server was closed or the customer blocked the network access to your application, so you can do the actions that you consider.

Advanced Options

There are some specific advanced options that can be used when working with Network Instances. Please, refer to the Advanced Options panel 67 for more information.

Network Instances on protected DLLs

When applying network instances on a protected DLL, the protection cannot determine in boot time (before your DllMain is executed) the number of instances running. This is because the server/client instance is not fully registered until your DllMain is executed.

As the protection boot loader cannot determine the number of running clients until your DllMain is executed, you should call the function WLRegNetInstancesGet 167 to know the number of running instances and compare it with the maximum number of network instances allowed (WLRegNetInstancesMax 168). You have to call those functions after your DllMain has been executed.

1.7 WinLicense SDK

The WinLicense SDK allows Software Developers to interact with WinLicense and retrieve/set information to manage your licenses/trial periods, create license keys, etc.

To be able to use the WinLicense API, an application must link with the Winlicense SDK library (*WinlicenseSDK lib/dll*).

NOTE: The *WinlicenseSDK.dll* is needed at protection time only. **The WinlicenseSDK.dll must not be released with your protected program**.

NOTE: For .NET applications, if you are building a 64-bit application (adding platform "x64") you have to define the conditional symbol **WIN64** (In Visual Studio, go to your project properties, select "Build" and enter the conditional symbol)

The different types of functions in the WinLicense SDK are the followings:

- **Trial Functions** 106
- **Registration Functions** 137
- **Generators Functions** 194
- **Miscellaneous Functions** 236

1.7.1 Trial Functions

The following functions can be used to handle the trial period in an application.

Function	Description
WLTrialCustomCounter 107	Retrieves the value of a specific counter
WLTrialCustomCounter-Dec 108	Decrements a custom counter by a specific value
WLTrialCustomCounterInc 110	Increments a custom counter by a specific value
WLTrialCustomCounterSet 111	Sets a specific value to a custom counter
WLTrialDateDaysLeft 112	Retrieves the number of trial days left when you are protecting with the option **Date Expiration**
WLTrialDaysLeft 114	Retrieves the number of trial days left before expiring.
WLTrialDebugCheck 115	Checks if the application was protected with the option "Trial Debug Mode" enabled
WLTrialExecutionsLeft 116	Retrieves the number of trial executions left before expiring.
WLTrialExpirationDate 117	Retrieves the trial expiration date in an application.
WLTrialExpirationTimestamp 118	Retrieves the exact expiration timestamp when an application is protected with ***trial days expiration***
WLTrialExpireTrial 119	Expires the trial period in the current application
WLTrialExtendExpiration 120	Allows extending the current trial period.
WLTrialExtGetLevel 121	Retrieves the current trial extension level.
WLTrialExtGetStatus 122	Retrieves information about a possible trial extension in the current application.

WLTrialFirstRun 123	Determines if an application is running in trial mode for the first time.
WLTrialGetTrialRestrictions 124	Retrieves a bit mask with the trial restrictions included in the protected application.
WLTrialGetStatus 126	Retrieves the trial status in the current application.
WLTrialGlobalTimeLeft 127	Retrieves the number of minutes left an application can be executed before expiring.
WLTrialLockedCountry 128	Retrieves the country code in which an application can be executed in trial mode.
WLTrialRuntimeLeft 129	Retrieves the runtime left an application can be running in memory.
WLTrialStringRead 129	Reads a custom string from the Registry.
WLTrialStringReadW 131	Reads a custom string (in UNICODE) from the Registry.
WLTrialStringWrite 132	Writes a custom string to the Registry.
WLTrialStringWriteW 134	Writes a custom string (in UNICODE) to the Registry.
WLTrialTotalDays 135	Retrieves the total number of trial days an application can be executed.
WLTrialTotalExecutions 136	Retrieves the total number of trial executions an application can be executed.

1.7.1.1 WLTrialCustomCounter

The **WLTrialCustomCounter** function retrieves the value of a specific counter. This is a general-purpose function used by applications that need a global counter to maintain the count of an expirable resource. For example, it can be used in an application that allows sending a maximum of 20 emails in trial period. Call this function to know how many emails have been sent.

Show C/C++ function definition

```
int WLTrialCustomCounter(
     int CounterID
);
```

Show Delphi function definition

```
function WLTrialCustomCounter(
     CounterID:Integer
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialCustomCounter Lib "WinLicenseSDK.dll" (
     ByVal CounterID As Integer
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialCustomCounter(int CounterId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialCustomCounter(CounterId As Integer) As Integer
End Function
```

Parameters

CounterID

This parameter specifies the counter ID to retrieve the value from.

The current version of WinLicense allows 3 different IDs (0, 1, 2). Each *CounterID* has its own local counter.

Return Values

If the function succeeds, the return value is the current value of the specified counter.

If the function fails, the return value is -1 (Invalid counter ID).

See Also

WLTrialCustomCounterInc 110, WLTrialCustomCounterDec 108, WLTrialCustomCounter-Set 111

1.7.1.2 WLTrialCustomCounterDec

The **WLTrialCustomCounterDec** function decrements a custom counter by a specific value. This is a general-purpose function that can be used by applications that needs a global counter to maintain the count of an expirable resource.

Show C/C++ function definition

```
int WLTrialCustomCounterDec(
     int Value,
     int CounterID
);
```

Show Delphi function definition

```
function WLTrialCustomCounterDec(
     Value:Integer;
     CounterID:Integer
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialCustomCounterDec Lib "WinLicenseSDK.dll" (
     ByVal Value As Integer,
     ByVal CounterID As Integer
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialCustomCounterDec(int Value, int CounterId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialCustomCounterDec(Value As Integer, CounterId As
     Integer) As Integer
End Function
```

Parameters

Value

This parameter specifies the value to decrement a specific counter ID.

CounterID

This parameter specifies the counter ID to decrement.

The current version of WinLicense allows 3 different IDs (0, 1, 2). Each *CounterID* has its own local counter.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is *wlInvalidCounter*.

See Also

[WLTrialCustomCounterInc](110), [WLTrialCustomCounter](107), [WLTrialCustomCounterSet](111)

1.7.1.3 WLTrialCustomCounterInc

The **WLTrialCustomCounterInc** function increments a custom counter by a specific value. This is a general-purpose function that can be used by applications that needs a global counter to maintain the count of an expirable resource. For example, it can be used in an application that allows sending a maximum of 20 emails in trial period. Calling this function every time that an email has been sent.

Show C/C++ function definition

```
int WLTrialCustomCounterInc(
     int Value,
     int CounterID
);
```

Show Delphi function definition

```
function WLTrialCustomCounterInc(
     Value:Integer;
     CounterID:Integer
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialCustomCounterInc Lib "WinLicenseSDK.dll" (
     ByVal Value As Integer,
     ByVal CounterID As Integer
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialCustomCounterInc(int Value, int CounterId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialCustomCounterInc(Value As Integer, CounterId As
     Integer) As Integer
End Function
```

Parameters

Value

This parameter specifies the value to increment a specific counter ID.

CounterID

This parameter specifies the counter ID to increment.

The current version of WinLicense allows 3 different IDs (0, 1, 2). Each *CounterID* has its own local counter.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is *wlInvalidCounter*.

See Also

WLTrialCustomCounterDec 108, WLTrialCustomCounter 107, WLTrialCustomCounterSet 111

1.7.1.4 WLTrialCustomCounterSet

The **WLTrialCustomCounterSet** function sets a specific value in a counter. This is a general-purpose function that can be used by applications that needs a global counter to maintain the count of an expirable resource.

Show C/C++ function definition

```
int WLTrialCustomCounterSet(
     int Value,
     int CounterID
);
```

Show Delphi function definition

```
function WLTrialCustomCounterSet(
     Value:Integer;
     CounterID:Integer
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialCustomCounterSet Lib "WinLicenseSDK.dll" (
     ByVal Value As Integer,
     ByVal CounterID As Integer
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialCustomCounterSet(int Value, int CounterId);

[Visual Basic]
```

```
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialCustomCounterSet(Value As Integer, CounterId As
     Integer) As Integer
End Function
```

Parameters

Value

This parameter specifies the value to set in counter.

CounterID

This parameter specifies the counter ID to set.

The current version of WinLicense allows 3 different IDs (0, 1, 2). Each *CounterID* has its own local counter.

Return Values

If the function succeeds, the return value is zero.

If the function fails, the return value is *wlInvalidCounter*.

See Also

WLTrialCustomCounterDec 108, WLTrialCustomCounter 107, WLTrialCustomCounter-Inc 110

1.7.1.5 WLTrialDateDaysLeft

The **WLTrialDateDaysLeft** function retrieves the number of trial days left when you are protecting with the option **Date Expiration**.

Show C/C++ function definition

```
int WLTrialDateDaysLeft(void);
```

Show Delphi function definition

```
function WLTrialDateDaysLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialDateDaysLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialDateDaysLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialDateDaysLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of trial days left before expiring.

If the current application does not have trial expiration date, the return value is *wlNoTrialDate (-1)*.

Remarks

Please, notice that this function returns the number of days left when you enable the trial **Date Expiration** option. If you are protecting your application with **Days Expiration** and you want to know the number of days left, you have to use the function WLTrialDaysLeft 114.

1.7.1.6 WLTrialDaysLeft

The **WLTrialDaysLeft** function retrieves the number of trial days left before expiring.

Show C/C++ function definition

```
int WLTrialDaysLeft(void);
```

Show Delphi function definition

```
function WLTrialDaysLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialDaysLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialDaysLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialDaysLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of trial days left before expiring.

If the trial has been manipulated by a user or the function is called without adding the **Days Expiration** option to the protected application, the return value is **-1**.

Remarks

Before calling WLTrialDaysLeft, you should call the function WLTrialGetStatus 126, to know the status of the trial (expired, manipulated, etc).

Please, notice that this function returns the number of days left when you enable the trial **Days Expiration** option. If you are protecting your application with **Date Expiration** and you want to know the number of days left, you have to use the function WLTrialDateDaysLeft 112.

See Also

WLTrialTotalDays 135

1.7.1.7 WLTrialDebugCheck

The **WLTrialDebugCheck** function checks if the application was protected with the option "Trial Debug Mode" enabled. This function allows you to check from your application that you are not releasing an application protected in "Trial Debug Mode" by mistake.

Show C/C++ function definition

```
     bool WLTrialDebugCheck();
```

Show Delphi function definition

```
function WLTrialDebugCheck():Boolean;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialDebugCheck Lib "WinLicenseSDK.dll" () As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLTrialDebugCheck();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialDebugCheck() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the application was protected with option "Trial Debug Mode" checked, the return value is *True*.

If the application was protected with option "Trial Debug Mode" unchecked, the return value is *False*.

1.7.1.8 WLTrialExecutionsLeft

The **WLTrialExecutionsLeft** function retrieves the number of trial executions left before expiring.

Show C/C++ function definition

```
int WLTrialExecutionsLeft(void);
```

Show Delphi function definition

```
function WLTrialExecutionsLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialExecutionsLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialExecutionsLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialExecutionsLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of trial executions left before expiring.

If the application was protected without Trial executions, the return value is *-1*.

See Also

WLTrialTotalExecutions 136

1.7.1.9 WLTrialExpirationDate

The **WLTrialExpirationDate** function retrieves the trial expiration date in an application.

Show C/C++ function definition

```
int WLTrialExpirationDate(
     SYSTEMTIME* pExpDate
);
```

Show Delphi function definition

```
function WLTrialExpirationDate(
     var pExpDate:SYSTEMTIME
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialExpirationDate Lib "WinLicenseSDK.dll" (
     pExpDate As Any
) As Integer
```

Show .NET function definition

```
[C#]
Kernel32.GetEnvironmentVariable("WLTrialExpirationDate", [out] ExpirationDate,
     100);

[Visual Basic]
GetEnvironmentVariable("WLTrialExpirationDate", [out] ExpirationDate, 100)
```

Parameters

pExpDate

[out] Pointer to a SYSTEMTIME structure that receives the trial expiration date.

Return Values

If the function success the return value is zero.

If the current application does not have trial expiration date, the return value is *wlNoTrialDate (-1)*.

Remarks

Before calling WLTrialExpirationDate, you should call the function WLTrialGetStatus 126, to know the status of the trial (expired, manipulated, etc).

1.7.1.10 WLTrialExpirationTimestamp

The **WLTrialExpirationTimestamp** function gets the exact expiration timestamp when an application is protected with *trial days expiration*. This function can be called when an application has been protected with trial days expiration to know the exact date and time that the trial will expire.

Show C/C++ function definition

```
bool WLTrialExpirationTimestamp(
     FILETIME* pFileTime
);
```

Show Delphi function definition

```
function WLTrialExpirationTimestamp(
     var pFileTime:FILETIME
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialExpirationTimestamp Lib "WinLicenseSDK.dll" (
     pExpDate As Any
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLTrialExpirationTimestamp(ref System.Runtime.In-
     teropServices.ComTypes.FILETIME ExpDate);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialExpirationTimestamp(ByRef ExpDate As Sys-
     tem.Runtime.InteropServices.ComTypes.FILETIME) As Boolean
End Function
```

Parameters

pFileTime
[out] Pointer to a FILETIME structure that receives the expiration timestamp.

Return Values

If the function success the return value is *True*.

If the current application is not protected with trial days expiration, the return value is *False*.

Remarks

This function only works when the application is protected with trial days expiration. In other cases, it will always return False.

1.7.1.11 WLTrialExpireTrial

The **WLTrialExpireTrial** function expires the trial period in the current application. This function is rarely called by some applications to abnormally finish the trial period when a critical situation has been found. When this function is called, the next time that an application is executed, its trial period will be expired.

Show C/C++ function definition

```
bool WLTrialExpireTrial(void);
```

Show Delphi function definition

```
function WLTrialExpireTrial():Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialExpireTrial Lib "WinLicenseSDK.dll" () As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLTrialExpireTrial();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialExpireTrial() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

1.7.1.12 WLTrialExtendExpiration

The **WLTrialExtendExpiration** function allows extending the current trial period. To extend the trial period in an application, we recommend using Trial Extension keys (Generated by WinLicense). The **WLTrialExtendExpiration** function is suitable for those applications which cannot accept WinLicense trial extension keys by design.

Show C/C++ function definition

```
bool WLTrialExtendExpiration(
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpDate,
     int Runtime,
     int GlobalTime
););
```

Show Delphi function definition

```
function WLTrialExtendExpiration(
     NumDays: Integer;
     NumExec: Integer;
     var pExpDate: SYSTEMTIME;
     Runtime: Integer;
     GlobalTime: Integer
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialExtendExpiration Lib "WinLicenseSDK.dll" (
     ByVal NumDays As Integer,
     ByVal NumExec As Integer,
     pExpDate As Any,
     ByVal Runtime As Integer,
     ByVal GlobalTime As Integer
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLTrialExtendExpiration(int NumDays, int NumExec,
     SystemTime ExpDate, int Runtime, int GlobalTime);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialExtendExpiration(NumDays As Integer, NumExec As
     Integer, ExpDate As SystemTime, Runtime As Integer,
            GlobalTime As Integer) As Boolean
End Function
```

Parameters

NumDays
This parameter specifies the number of days to extend the trial.

NumExec
This parameter specifies the number of executions to extend the trial.

pExpDate
This parameter specifies a pointer to a **SYSTEMTIME** structure with the new expiration date.

Runtime
This parameter specifies the new runtime expiration to extend in trial mode.

GlobalTime
This parameter specifies the number of minutes to extend the usage of the protected application in trial mode.

Return Values

If the function succeeds, the return value is true.

If the function fails, the return value is *false*.

1.7.1.13 WLTrialExtGetLevel

The **WLTrialExtGetLevel** function retrieves the current trial extension level.

Show C/C++ function definition

```
int WLTrialExtGetLevel(void);
```

Show Delphi function definition

```
function WLTrialExtGetLevel():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialExtGetLevel Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialExtGetLevel();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
```

```
    Public Shared Function WLTrialExtGetLevel() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the current trial extension level. If no trial extension is found, the return value is zero.

1.7.1.14 WLTrialExtGetStatus

The **WLTrialExtGetStatus** function retrieves information about a possible trial extension in the current application.

Show C/C++ function definition

```
int WLTrialExtGetStatus(void);
```

Show Delphi function definition

```
function WLTrialExtGetStatus():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialExtGetStatus Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
Kernel32.GetEnvironmentVariable("WLTrialExtGetStatus", [out] TrialExtStatus, 100);

[Visual Basic]
GetEnvironmentVariable("WLTrialExtGetStatus", [out] TrialExtStatus, 100)
```

Parameters

This function has no parameters.

Return Values

The return value is the trial extension status in the current application. The possible values are the followings:

- *wlNoTrialExt (0)* when the trial extension key not present.
- *wlAppExtended (1)* when the application is extended with trial extension key.
- *wlInvalidTrialExt (2)* when a trial extension key is invalid.
- *wlNoMoreExt (3)* when no more extension keys are allowed.

See Also

WLTrialGetStatus 126

1.7.1.15 WLTrialFirstRun

The **WLTrialFirstRun** function determines if an application is running in Trial mode for the first time.

Show C/C++ function definition

```
bool WLTrialFirstRun(void);
```

Show Delphi function definition

```
function WLTrialFirstRun():Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialFirstRun Lib "WinLicenseSDK.dll" () As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLTrialFirstRun();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialFirstRun() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the application is launched for the first time and it's running in Trial mode, the return value is *True*, otherwise it returns *False*.

See Also

WLRegFirstRun 153

1.7.1.16 WLTrialGetTrialRestrictions

The **WLTrialGetTrialRestrictions** function retrieves a bit mask with the trial restrictions included in the protected application, that is, the restrictions set in the "Trial Settings" panel in WinLicense.

Show C/C++ function definition

```
int WLTrialGetTrialRestrictions(void);
```

Show Delphi function definition

```
function WLTrialGetTrialRestrictions():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialGetTrialRestrictions Lib "WinLicenseSDK.dll" () As
Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialGetTrialRestrictions();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialGetTrialRestrictions() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is a bit mask with the trial restrictions included in the protected application. Current supported values:

Value	Meaning
wlTrialRestrictionUnlimited 0x0000	No Trial restrictions set or *Unlimited* option checked
wlTrialRestrictionDays 0x0001	Days expiration is enabled
wlTrialRestrictionExec 0x0002	Executions expiration is enabled
wlTrialRestrictionDate 0x0004	Date expiration is enabled
wlTrialRestrictionRuntime 0x0008	Runtime restriction is enabled
wlTrialRestrictionGlobalTime 0x0010	Global time restriction is enabled
wlTrialRestrictionCountry 0x0020	Country restriction is enabled

1.7.1.17 WLTrialGetStatus

The **WLTrialGetStatus** function retrieves the trial status in the current application.

Show C/C++ function definition

```
int WLTrialGetStatus(
     int* Reserved
);
```

Show Delphi function definition

```
function WLTrialGetStatus(
     var Reserved:Integer
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialGetStatus Lib "WinLicenseSDK.dll" (
     Reserved As Any
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialGetStatus(ref int Reserved);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialGetStatus(ByRef Reserved As Integer) As Integer
End Function
```

Parameters

Reserved
This parameter contains a special code with extra information about the status of the trial. This parameter is reserved in current versions of WinLicense.

Return Values

The return value is the trial status in the current application. The possible values are the followings:

- *wlTrialOk (0)* when trial OK.
- *wlTrialModeNotSupported (16)* when trial is not available (not selected in "Trial Settings" panel).

- *wlTrialDaysExpired (1)* when the trial days expired.
- *wlTrialExecExpired (2)* when the trial executions expired.
- *wlTrialDateExpired (3)* when the trial date expired.
- *wlTrialRuntimExpired (4)* when the runtime trial expired.
- *wlTrialGlobalExpired (5)* when the global trial time expired.
- *wlTrialInvalidCountry (6)* when the Trial does not run on the current language/country.
- *wlTrialManipulated (7)* when the clock has been turned back or a user has tried to cheat the trial period in your application.

See Also

WLTrialExtGetStatus 122, WLRegGetStatus 161

1.7.1.18 WLTrialGlobalTimeLeft

The **WLTrialGlobalTimeLeft** function retrieves the number of minutes left that an application can be executed before expiring.

Show C/C++ function definition

```
int WLTrialGlobalTimeLeft(void);
```

Show Delphi function definition

```
function WLTrialGlobalTimeLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialGlobalTimeLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialGlobalTimeLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialGlobalTimeLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of minutes left before expiring.

1.7.1.19 WLTrialLockedCountry

The **WLTrialLockedCountry** function retrieves the country code in which an application can be executed in trial mode.

Show C/C++ function definition

```
int WLTrialLockedCountry(void);
```

Show Delphi function definition

```
function WLTrialLockedCountry():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialLockedCountry Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialLockedCountry();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialLockedCountry() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the country code of the locked trial. To see a list of the possible country codes, check the country codes list.

1.7.1.20 WLTrialRuntimeLeft

The **WLTrialRuntimeLeft** function retrieves the runtime left that an application can be running in memory.

Show C/C++ function definition

```
int WLTrialRuntimeLeft(void);
```

Show Delphi function definition

```
function WLTrialRuntimeLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialRuntimeLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialRuntimeLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialRuntimeLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the runtime left in minutes.

1.7.1.21 WLTrialStringRead

The **WLTrialStringRead** function reads a custom string from the Registry. The string is stored using encryption algorithms and the location of the string is different for each computer.

The location of the string also depends on the "Trial Unique Key" in the protected software. If you change the "Trial Unique Key" in your Software and protect again, the strings will be stored in a different Registry location.

NOTE: If StringName starts with "!", the custom string will be stored under HKEY_LOCAL_MACHINE, otherwise it will be stored under HKEY_CURRENT_USER in the Windows Registry.

Show C/C++ function definition

```
bool WLTrialStringRead(
     char* pStringName,
     char* pStringValue
);
```

Show Delphi function definition

```
function WLTrialStringRead(
     pStringName:PAnsiChar;
     pStringValue:PAnsiChar;
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialStringRead Lib "WinLicenseSDK.dll" (
     ByVal pStringName As String,
     ByVal pStringValue As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Ansi, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLTrialStringRead(string StringName, StringBuilder
     StringValue);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Ansi, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLTrialStringRead(StringName As String, ByRef
     StringValue As String) As Boolean
End Function
```

Parameters

pStringName
[in] Name of the string to be accessed.

pStringValue
[out] Value of the string stored under StringName.

Return Values

If the function succeeds, the return value is *True*.

If the StringName does not exist, the return value is *False*. Notice that if you have a value stored under "StringName" and you change the Trial Unique Key, the location of "StringName" will be different, hence this function will return *False* as the location for "StringName" is empty.

See Also

WLTrialStringWrite 132

1.7.1.22 WLTrialStringReadW

The **WLTrialStringReadW** function reads a custom string from the Registry. The string is stored using encryption algorithms and the location of the string is different for each computer.

The location of the string also depends on the "Trial Unique Key" in the protected software. If you change the "Trial Unique Key" in your Software and protect again, the strings will be stored in a different Registry location.

NOTE: If StringName starts with "!", the custom string will be stored under HKEY_LOCAL_MACHINE, otherwise it will be stored under HKEY_CURRENT_USER in the Windows Registry.

Show C/C++ function definition

```
bool WLTrialStringReadW(
     wchar_t* pStringName,
     wchar_t* pStringValue
);
```

Show Delphi function definition

```
function WLTrialStringReadW(
     pStringName:PWideChar;
     pStringValue:PWideChar;
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLTrialStringReadW(string StringName, StringBuilder
     StringValue);

[Visual Basic]
```

```
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLTrialStringReadW(StringName As String, ByRef
     StringValue As String) As Boolean
End Function)
```

Parameters

pStringName

[in] Name of the string to be accessed.

pStringValue

[out] Value of the string stored under StringName.

Return Values

If the function succeeds, the return value is *True*.

If the StringName does not exist, the return value is *False*. Notice that if you have a value stored under "StringName" and you change the Trial Unique Key, the location of "StringName" will be different, hence this function will return *False* as the location for "StringName" is empty.

See Also

WLTrialStringWriteW 134

1.7.1.23 WLTrialStringWrite

The **WLTrialStringWrite** function writes a custom string to the Registry. The string is stored using encryption algorithms and the location of the string is different for each computer.

The location of the string also depends on the "Trial Unique Key" in the protected software. If you change the "Trial Unique Key" in your Software and protect again, the strings will be stored in a different Registry location.

NOTE: If StringName starts with "!", the custom string will be stored under HKEY_LOCAL_MACHINE, otherwise it will be stored under HKEY_CURRENT_USER in the Windows Registry.

Show C/C++ function definition

```
bool WLTrialStringWrite(
     char* pStringName,
     char* pStringValue
);
```

Show Delphi function definition

```
function WLTrialStringWrite(
     pStringName:PAnsiChar;
     pStringValue:PAnsiChar;
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialStringWrite Lib "WinLicenseSDK.dll" (
     ByVal pStringName As String,
     ByVal pStringValue As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Ansi, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLTrialStringWrite(string StringName, string
     StringValue);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Ansi, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLTrialStringWrite(StringName As String, StringValue As
     String) As Boolean
End Function
```

Parameters

pStringName
[in] Name of the string to be accessed.

pStringValue
[in] Value of the string to be stored under StringName.

Return Values

If the function succeeds, the return value is *True*.

If the writing to the Registry fails, the return value is *False*.

See Also

WLTrialStringRead 129

1.7.1.24 WLTrialStringWriteW

The **WLTrialStringWriteW** function writes a custom string to the Registry. The string is stored using encryption algorithms and the location of the string is different for each computer.

The location of the string also depends on the "Trial Unique Key" in the protected software. If you change the "Trial Unique Key" in your Software and protect again, the strings will be stored in a different Registry location.

NOTE: If StringName starts with "**!**", the custom string will be stored under HKEY_LOCAL_MACHINE, otherwise it will be stored under HKEY_CURRENT_USER in the Windows Registry.

Show C/C++ function definition

```
bool WLTrialStringWriteW(
     wchar_t* pStringName,
     wchar_t* pStringValue
);
```

Show Delphi function definition

```
function WLTrialStringWriteW(
     pStringName:PWideChar;
     pStringValue:PWideChar;
):Boolean; stdcall
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLTrialStringWriteW(string StringName, string
     StringValue);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLTrialStringWriteW(StringName As String, StringValue As
     String) As Boolean
End Function
```

Parameters

pStringName
[in] Name of the string to be accessed.

pStringValue
[in] Value of the string to be stored under StringName.

Return Values

If the function succeeds, the return value is *True*.

If the writing to the Registry fails, the return value is *False*.

See Also

WLTrialStringReadW 131

1.7.1.25 WLTrialTotalDays

The **WLTrialTotalDays** function retrieves the total number of trial days that an application can be executed.

Show C/C++ function definition

```
int WLTrialTotalDays(void);
```

Show Delphi function definition

```
function WLTrialTotalDays():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialTotalDays Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialTotalDays();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialTotalDays() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of total trial days that an application can be executed.

See Also

WLTrialDaysLeft 114

1.7.1.26 WLTrialTotalExecutions

The **WLTrialTotalExecutions** function retrieves the total number of trial executions that an application can be executed.

Show C/C++ function definition

```
int WLTrialTotalExecutions(void);
```

Show Delphi function definition

```
function WLTrialTotalExecutions():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLTrialTotalExecutions Lib "WinLicenseSDK.dll" () As In-
teger
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLTrialTotalExecutions();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLTrialTotalExecutions() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of total trial executions that an application can be executed.

See Also

WLTrialExecutionsLeft 116

1.7.2 Registration Functions

The following functions can be used to handle the registration information in an application.

Function	*Description*
WLRegActivateSoftware 139	Performs activation of an application from a given activation key.
WLRegCheckMachineLocked 142	Determines if the current license is locked to a specific machine.
WLRegDateDaysLeft 142	Retrieves the number of days left in the current license when the license has **Date Expiration**.
WLRegDaysLeft 144	Retrieves the number of days left in the current license key.
WLRegDeactivateSoftware 144	Performs deactivation of an application from a given activation key.
WLRegDisableCurrentKey 147	Marks the current license key as stolen or invalid.
WLRegDisableKeyCurrentInstance 148	Disables the current installed key for the specific application which calls the function.
WLRegExecutionsLeft 149	Retrieves the number of executions left in the current license key.
WLRegExpirationDate 151	Retrieves the expiration date in a license key.
WLRegExpirationTimestamp 152	Retrieves the exact expiration timestamp for a license key with **days expiration**.
WLRegFirstRun 153	Determines if an application is just registered with a valid license key.
WLRegGetDynSmartKey 154	Retrieves the Dynamic SmartActivate key that was inserted to register an application.
WLRegGetLicenseHardwareID 155	Retrieves the hardware ID in the current license key.
WLRegGetLicenseInfo 156	Retrieves the registration information for the current license.
WLRegGetLicenseInfoW 157	Retrieves the registration information (in UNICODE) for the current license.
WLRegGetLicenseRestrictions 158	Retrieves the restrictions included in the current license.
WLRegGetStatus 161	Retrieves information about the licensing status in the current application.

WLRegGlobalTimeLeft 163	Retrieves the number of minutes left in the current license key.
WLRegLicenseCreationDate 164	Retrieves the creation date in a license key.
WLRegLicenseName 165	Retrieves the license file and registry names for the expected registration license.
WLRegLockedCountry 166	Retrieves the country code in the current license key.
WLRegNetInstancesGet 167	Retrieves the current number of instances running in a network.
WLRegNetInstancesGetClientsIp 168	Retrieves the IP addresses of all running clients.
WLRegNetInstancesMax 168	Retrieves the maximum number of instances allowed in the current license.
WLRegNormalKeyCheck 169	Validates a text key.
WLRegNormalKeyCheckW 171	Validates a text key (UNICODE).
WLRegNormalKeyInstallToFile 172	Installs a license text key into a file.
WLRegNormalKeyInstallToFileW 173	Installs a license text key into a file (UNICODE).
WLRegNormalKeyInstallToRegistry 174	Installs a license text key into a Windows registry key.
WLRegNormalKeyInstallToRegistryW 175	Installs a license text key into a Windows registry key (UNICODE).
WLRegRemoveCurrentKey 176	Removes the current license key from the system.
WLRegRuntimeLeft 177	Retrieves the runtime left that a registered application can be running in memory.
WLRegSmartKeyCheck 178	Validates a SmartActivate® key.
WLRegSmartKeyCheckW 180	Validates a SmartActivate® key (UNICODE).
WLRegSmartKeyInstallToFile 182	Installs a SmartActivate® key into a file.
WLRegSmartKeyInstallToFileW 183	Installs a SmartActivate® key into a file (UNICODE).
WLRegSmartKeyInstallToFileInFolder 185	Installs a SmartActivate® key into a file in a specific folder.
WLRegSmartKeyInstallToFileInFolderW 187	Installs a SmartActivate® key into a file in a specific folder (UNICODE).

WLRegSmartKeyInstallToRegistry 189	Installs a SmartActivate® key into a Windows registry key.
WLRegSmartKeyInstallToRegistryW 191	Installs a SmartActivate® key into a Windows registry key (UNICODE).
WLRegTotalDays 193	Retrieves the total number of days in the current license key.
WLRegTotalExecutions 193	Retrieves the total number of executions in the current license key.

1.7.2.1 WLRegActivateSoftware

The **WLRegActivateSoftware** function performs activation of your application from a given activation key. Internally, the function calls the activation PHP page in your web server (defined in the Activation panel) and it should receive a license key (locked to the current machine) if the activation key is valid.

Show C/C++ function definition

```
bool WLRegActivateSoftware (
     char* pActivationKey,
     int*  pWinsockErrorCode,
     char* pServerResponseBuffer,
     int   SizeServerResponseBuffer);
```

Show Delphi function definition

```
function WLRegActivateSoftware (
     pActivationKey: PAnsiChar;
     pWinsockErrorCode: PAnsiChar;
     pServerResponseBuffer: PAnsiChar;
     SizeServerResponseBuffer: Integer;
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegActivateSoftware Lib "WinLicenseSDK.dll" (
     ByVal pActivationKey As String,
     ByVal pWinsockErrorCode As Any,
     ByVal pServerResponseBuffer As String,
     ByVal SizeServerResponseBuffer As Integer,
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegActivateSoftware(string ActivationKey, ref int
     WinsockErrorCode, StringBuilder ServerResponseBuffer,
            int SizeServerResponseBuffer);
```

```
[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegActivateSoftware(ActivationKey As String, ByRef
     WinsockErrorCode As Integer, ByRef ServerResponseBuffer As String,
            SizeServerResponseBuffer As Integer) As Boolean
End Function
```

Parameters

pActivationKey
[in] Pointer to an ASCII string that will contain the activation key

WinsockErrorCode
[out] This parameter contains the Winsock error code in case that you get the "RESPONSE_ERROR_WINSOCK_ERROR" return value

ServerResponseString
[out] This parameter contains the buffer returned by your PHP activation page. You might want to display this buffer to the customer so he can send you further information when the activation key is not processed correctly.

SizeServerResponseString
[in] This parameter contains the size of the *ServerResponseString* buffer

Return Values

The possible return values are:

RESPONSE_ACTIVATION_OK (0)	The activation key was processed correctly and a license has been retrieved from the server to register your application
RESPONSE_ERROR_KEY_NOT_FOUND (1)	The entered activation key is not found in the database
RESPONSE_ERROR_MAX_SIMULTANEOUS_ACTIVATIONS_REACHED (2)	The activation key is currently activating the maximum number of simultaneous devices. The maximum number of simultaneous devices is specified by the "Simultaneous Devices" option in "WL Orders Manager" (in the "Manage Activations" panel). If the customer wants to activate a new device, he first needs to deactivate one active device.
RESPONSE_ERROR_NO_MORE_ACTIVATIO	The maximum number of activation for the current Activation key has been reached. You can control the maximum

NS_ALLOWED (3)	number of activations from the "Activations Limit" option in "WL Orders Manager" (in the "Manage Activations" panel)
RESPONSE_ERROR_DEVICE_NOT_FOUND (5)	The device to deactivate is not found in the database.
RESPONSE_ERROR_WRONG_DATA_RECEIVED (6)	The activation key is processed correctly but the license data is not retrieved correctly (the license generator application is not generating an expected output)
RESPONSE_ERROR_KEY_DISABLED_BY_SELLER (7)	The activation key has been manually marked as "Disabled" in "WL Orders Manager" (in "Manage Activations" panel)
RESPONSE_ERROR_KEY_EXPIRED (8)	The expiration date (to allow activation) is already reached. This expiration date was inserted when the activation key was created in "WL Orders Manager". You can manually extend this date if required from the "Manage Activations" panel in "WL Orders Manager"
RESPONSE_ERROR_NO_MORE_DIFFERENT_DEVICES_ALLOWED (9)	The activation code has been already activated in more different PCs that were allowed. You can control the maximum number of different devices from the "Max. Different Devices" option in "WL Orders Manager" (in the "Manage Activations" panel)
RESPONSE_ERROR_CANNOT_INSTALL_LICENSE (50)	The activation key was processed correctly and the server sent back a correct license key, but it was not possible to write it to the expected location in the customer's PC. This happens if your application does not have rights to write to the expected location for your licenses (you can set the location of your license in the "Registration 36" panel)
RESPONSE_ERROR_WINSOCK_ERROR (100)	This error appears when there are problems to access to the internet or your web server from your application. To know the exact Winsock error code you can read the "WinsockErrorCode" parameter

See Also

WLRegDeactivateSoftware 144

1.7.2.2 WLRegCheckMachineLocked

The **WLRegCheckMachineLocked** function determines if the current license is locked to a specific machine.

Show C/C++ function definition

```
bool WLRegCheckMachineLocked(void);
```

Show Delphi function definition

```
function WLRegCheckMachineLocked():Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegCheckMachineLocked Lib "WinLicenseSDK.dll" () As
Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegCheckMachineLocked();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegCheckMachineLocked() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the current license key is locked to a specific machine, the return value is *True*.

If the application is not licensed or the current license is not locked to a specific machine, the return value is *False*.

1.7.2.3 WLRegDateDaysLeft

The **WLRegDateDaysLeft** function retrieves the number of days left in the current license when the license has **Date Expiration**.

Show C/C++ function definition

```
int WLRegDateDaysLeft(void);
```

Show Delphi function definition

```
function WLRegDateDaysLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegDateDaysLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegDateDaysLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegDateDaysLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of days left in the current license key.

Remarks

Please, notice that this function returns the number of days left when the license has **Date Expiration**. If the license has **Days Expiration** and you want to know the number of days left, you have to use the function WLRegDaysLeft 144.

1.7.2.4 WLRegDaysLeft

The **WLRegDaysLeft** function retrieves the number of days left in the current license key.

Show C/C++ function definition

```
int WLRegDaysLeft(void);
```

Show Delphi function definition

```
function WLRegDaysLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegDaysLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegDaysLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegDaysLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of days left in the current license key.

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

See Also

WLRegTotalDays 193

1.7.2.5 WLRegDeactivateSoftware

The **WLRegDeactivateSoftware** function performs deactivation of your application from a given activation key. Internally, the function calls the activation PHP page in your web server (defined in the Activation panel) and if the activation key is valid, the license will be removed (and invalidated) from the current system.

Show C/C++ function definition

```
bool WLRegDectivateSoftware (
     char* pActivationKey,
     int*  pWinsockErrorCode,
     char* pServerResponseBuffer,
     int   SizeServerResponseBuffer);
```

Show Delphi function definition

```
function WLRegDectivateSoftware (
     pActivationKey: PAnsiChar;
     pWinsockErrorCode: PAnsiChar;
     pServerResponseBuffer: PAnsiChar;
     SizeServerResponseBuffer: Integer;
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegDectivateSoftware Lib "WinLicenseSDK.dll" (
     ByVal pActivationKey As String,
     ByVal pWinsockErrorCode As Any,
     ByVal pServerResponseBuffer As String,
     ByVal SizeServerResponseBuffer As Integer,
) As Boolean
```

Show .NET function definition

```
[C#]
Kernel32.GetEnvironmentVariable("WLRegDectivateSoftware", [out] ActivationInfo,
     3000);

[Visual Basic]
GetEnvironmentVariable("WLRegDectivateSoftware", [out] ActivationInfo, 3000)
```

Parameters

pActivationKey
    [in] Pointer to an ASCII string that will contain the activation key

WinsockErrorCode
    [out] This parameter contains the Winsock error code in case that you get the "RESPONSE_ERROR_WINSOCK_ERROR" return value

ServerResponseString

[out] This parameter contains the buffer returned by your PHP activation page. You might want to display this buffer to the customer so he can send you further information when the activation key is not processed correctly.

SizeServerResponseString
[in] This parameter contains the size of the *ServerResponseString* buffer

Return Values

The possible return values are:

RESPONSE_DEACTIVATION_OK (0)	The deactivation process went OK and the current license has been removed from the system and marked as banned. The current PC is marked as revoked in the activation database
RESPONSE_ERROR_KEY_NOT_FOUND (1)	The entered activation key is not found in the database
RESPONSE_ERROR_NO_MORE_DEACTIVATIONS_ALLOWED (4)	The customer has deactivated more devices than allowed. You can control the maximum number of deactivations from the "Deactivation Limit" option in "WL Orders Manager" (in the "Manage Activations" panel)
RESPONSE_ERROR_DEVICE_NOT_FOUND (5)	The device to deactivate is not found in the database.
RESPONSE_ERROR_WRONG_DATA_RECEIVED (6)	The activation key is processed correctly but the license data is not retrieved correctly (the license generator application is not generating an expected output)
RESPONSE_ERROR_KEY_DISABLED_BY_SELLER (7)	The activation key has been manually marked as "Disabled" in "WL Orders Manager" (in "Manage Activations" panel)
RESPONSE_ERROR_CANNOT_INSTALL_LICENSE (50)	The activation key was processed correctly and the server sent back a correct license key, but it was not possible to write it to the expected location in the customer's PC. This happens if your application does not have rights to write to the expected location for your licenses (you can set the location of your license in the "Registration 36" panel)

RESPONSE_ERROR_ WINSOCK_ERROR (100)	This error appears when there are problems to access to the internet or your web server from your application. To know the exact Winsock error code you can read the "WinsockErrorCode" parameter

See Also

WLRegActivateSoftware 139

1.7.2.6 WLRegDisableCurrentKey

The **WLRegDisableCurrentKey** function marks the current license key as stolen or invalid. This function can be used when an application, with internet access, detects that the current key has been stolen. Winlicense will block the license key in runtime and will prevent the application from running again with that license key.

Show C/C++ function definition

```
bool WLRegDisableCurrentKey (
     int DisableFlags
);
```

Show Delphi function definition

```
function WLRegDisableCurrentKey (
     DisableFlags:Integer
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegDisableCurrentKey  Lib "WinLicenseSDK.dll" (
     ByVal DisableFlags As Integer
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegDisableCurrentKey(int DisableFlags);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegDisableCurrentKey(DisableFlags As Integer) As
     Boolean
End Function
```

Parameters

DisableFlags

This parameter specifies the status to set up in the current license key.

The current version of WinLicense allows the following values for *DisableFlags*:

- *wlMarkStolenKey (0)* to mark a key as stolen.
- *wlMarkInvalidKey (1)* to mark a key as invalid.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

Remarks

When you call WLRegDisableCurrentKey, basically it bans the specific user information ("User Name/Company/Custom Data") for your application. If you re-generate a new license with the same user information, it will be still invalid.

You have to re-generate a new license but changing any user information (like for example, insert/remove some extra chars in the "Custom Data" field)

See Also

WLRegRemoveCurrentKey 176

1.7.2.7 WLRegDisableKeyCurrentInstance

The **WLRegDisableKeyCurrentInstance** function disables the current installed key for the specific application which calls **WLRegDisableKeyCurrentInstance**. That is, if you have 2 versions of your application and one of them calls WLRegDisableKeyCurrentInstance, then the current installed license will not be valid for that specific instance (the "*MsgID42 : License disabled in current instance*" from Customized Dialogs 43 will be displayed). Your other protected version will work fine with the current installed license.

A typical case example is when you sell a product and your customers need to pay for a subscription after some time. So, your customers will have access to your latest release versions. When the subscription expires for a customer (for example, you store the subscription expiration date in the "Custom Data" inside a license), you call **WLRegDisableKeyCurrentInstance** and the license will not be valid for that specific version which called that function. If your customer runs previous versions of your application, the license will be recognized and your customer can keep using previous versions of your application, but he won't be able to run the version which called **WLRegDisableKeyCurrentInstance**.

Show C/C++ function definition

```
bool WLRegDisableKeyCurrentInstance();
```

Show Delphi function definition

```
function WLRegDisableKeyCurrentInstance():Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegDisableKeyCurrentInstance  Lib "WinLicenseSDK.dll" ()
     As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegDisableKeyCurrentInstance();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegDisableKeyCurrentInstance() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the license was successfully disabled, the return value is *True*.

If the function fails, the return value is *False*.

1.7.2.8 WLRegExecutionsLeft

The **WLRegExecutionsLeft** function retrieves the number of executions left in the current license key.

Show C/C++ function definition

```
int WLRegExecutionsLeft(void);
```

Show Delphi function definition

```
function WLRegExecutionsLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegExecutionsLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegExecutionsLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegExecutionsLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of executions left in the current license key.

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

See Also

WLRegTotalExecutions 193

1.7.2.9 WLRegExpirationDate

The **WLRegExpirationDate** function retrieves the expiration date in a license key.

Show C/C++ function definition

```
int WLRegExpirationDate(
     SYSTEMTIME* pExpDate
);
```

Show Delphi function definition

```
function WLRegExpirationDate(
     var pExpDate:SYSTEMTIME
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegExpirationDate Lib "WinLicenseSDK.dll" (
     pExpDate As Any
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegExpirationDate(SystemTime ExpDate);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegExpirationDate(ExpDate As SystemTime) As Integer
End Function
```

Parameters

pExpDate
[out] Pointer to a SYSTEMTIME structure that receives the license expiration date.

Return Values

If the function succeeds the return value is zero.

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

1.7.2.10 WLRegExpirationTimestamp

The **WLRegExpirationTimestamp** function gets the exact expiration timestamp for a license key with **days expiration**. This function can be called when a license with days expiration is present to know the exact date and time that the license will expire.

Show C/C++ function definition

```
bool WLRegExpirationTimestamp(
     FILETIME* pFileTime
);
```

Show Delphi function definition

```
function WLRegExpirationTimestamp(
     var pFileTime:FILETIME
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegExpirationTimestamp Lib "WinLicenseSDK.dll" (
     pExpDate As Any
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegExpirationTimestamp(ref System.Runtime.In-
     teropServices.ComTypes.FILETIME ExpDate);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegExpirationTimestamp(ByRef ExpDate As Sys-
     tem.Runtime.InteropServices.ComTypes.FILETIME) As Boolean
End Function
```

Parameters

pFileTime

[out] Pointer to a FILETIME structure that receives the expiration timestamp.

Return Values

If the function success the return value is *True*.

If the current application is not registered with a license with expiration days, the return value is *False*.

Remarks

This function only works with licenses with expiration days. In other cases, it will always return False.

1.7.2.11 WLRegFirstRun

The **WLRegFirstRun** function determines if an application is running in Registered mode for the first time.

Show C/C++ function definition

```
bool WLRegFirstRun(void);
```

Show Delphi function definition

```
function WLRegFirstRun():Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegFirstRun Lib "WinLicenseSDK.dll" () As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegFirstRun();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegFirstRun() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

When an application is just licensed, the return value is *True*. Notice that if an application is licensed again with a different license key, the return value will be *True*.

See Also

WLTrialFirstRun 123

1.7.2.12 WLRegGetDynSmartKey

The **WLRegGetDynSmartKey** retrieves the Dynamic SmartActivate key that was inserted by the customer to register a protected application.

Show C/C++ function definition

```
bool WLRegGetDynSmartKey (
     char* pDynSmartKey);
```

Show Delphi function definition

```
function WLRegGetDynSmartKey (
     pDynSmartKey:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegGetDynSmartKey  Lib "WinLicenseSDK.dll" (
     ByVal pDynSmartKey As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegGetDynSmartKey(StringBuilder DynSmartKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegGetDynSmartKey(ByRef DynSmartKey As String) As
     Boolean
End Function
```

Parameters

pDynSmartKey
[out] Pointer to a buffer that receives the registration Dynamic SmartActivate key.

Return Values

If the current license was registered with a Dynamic SmartActivate key, the return value is *True*.

If the current license was not registered with a Dynamic SmartActivate key, the return value is *False*.

1.7.2.13 WLRegGetLicenseHardwareID

The **WLRegGetLicenseHardwareID** retrieves the hardware ID in the current license key.

NOTE: This function is not available for SmartKeys due to length optimization in SmartKeys.

Show C/C++ function definition

```
bool WLRegGetLicenseHardwareID (
     char* pHardwareId);
```

Show Delphi function definition

```
function WLRegGetLicenseHardwareID (
     pHardwareId:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegGetLicenseHardwareID  Lib "WinLicenseSDK.dll" (
     ByVal pHardwareId As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegGetLicenseHardwareID(StringBuilder HardwareId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegGetLicenseHardwareID(ByRef HardwareId As String) As
     Boolean
End Function
```

Parameters

pHardwareId

[out] Pointer to a buffer that will receive a null-terminated string with the hardware ID in the current license key.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

Restrictions

This function does not work with Static or Dynamic Smartkeys.

See Also

WLHardwareGetID 248, **WLHardwareCheckID** 246, **WLHardwareGetFormattedID** 247

1.7.2.14 WLRegGetLicenseInfo

The **WLRegGetLicenseInfo** function retrieves the registration information for the current license.

Show C/C++ function definition

```
bool WLRegGetLicenseInfo(
     char* pName,
     char* pCompanyName,
     char* pCustomData
);
```

Show Delphi function definition

```
function WLRegGetLicenseInfo(
     pName:PAnsiChar;
     pCompanyName:PAnsiChar;
     pCustomData:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegGetLicenseInfo Lib "WinLicenseSDK.dll" (
     ByVal pName As String,
     ByVal pCompanyName As String,
     ByVal pCustomData As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegGetLicenseInfo(StringBuilder Name, StringBuilder
     CompanyName, StringBuilder CustomData);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegGetLicenseInfo(Name As StringBuilder, CompanyName
     As StringBuilder, CustomData As StringBuilder) As Boolean
End Function
```

Parameters

pName

[out] Pointer to a string that will receive the registration name in the current license key.

pCompanyName

[out] Pointer to a string that will receive the company name in the current license key.

pCustomData

[out] Pointer to a string that will receive the custom data in the current license key.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegGetStatus 161

1.7.2.15 WLRegGetLicenseInfoW

The **WLRegGetLicenseInfoW** function retrieves the registration information (in UNICODE) for the current license.

Show C/C++ function definition

```
bool WLRegGetLicenseInfoW(
     wchar_t* pName,
     wchar_t* pCompanyName,
     wchar_t* pCustomData
);
```

Show Delphi function definition

```
function WLRegGetLicenseInfoW(
     pName:PWideChar;
     pCompanyName:PWideChar;
     pCustomData:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
```

```
    public static extern bool WLRegGetLicenseInfoW(StringBuilder Name,
     StringBuilder CompanyName, StringBuilder CustomData);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegGetLicenseInfoW(ByRef Name As String, ByRef Com-
     panyName As String, ByRef CustomData As String) As Boolean
End Function
```

Parameters

pName
[out] Pointer to a string that will receive the registration name in the current license key.

pCompanyName
[out] Pointer to a string that will receive the company name in the current license key.

pCustomData
[out] Pointer to a string that will receive the custom data in the current license key.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegGetStatus 161

1.7.2.16 WLRegGetLicenseRestrictions

The WLRegGetLicenseRestrictions function retrieves a bit mask with the restrictions included in the current license. This function is suitable when you are registering your application with licenses with different registration restrictions (days expiration, executions, date expiration, etc) to know which WinLicense SDK functions you should call for each specific license.

Show C/C++ function definition

```
int WLRegGetLicenseRestrictions(void);
```

Show Delphi function definition

```
function WLRegGetLicenseRestrictions():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegGetLicenseRestrictions Lib "WinLicenseSDK.dll" () As
Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegGetLicenseRestrictions();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegGetLicenseRestrictions() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is a bit mask with the license restrictions included in the current license. Current supported values:

Value	Meaning
wlRegRestrictionDays 0x0001	Days expiration is enabled
wlRegRestrictionExec 0x0002	Executions expiration is enabled
wlRegRestrictionDate 0x0004	Date expiration is enabled
wlRegRestrictionRuntime 0x0008	Runtime restriction is enabled
wlRegRestrictionGlobalTime 0x0010	Global time restriction is enabled
wlRegRestrictionCountry 0x0020	Country restriction is enabled

wlRegRestrictionHardwareId 0x0040	License has hardware ID restriction
wlRegRestrictionNetwork 0x0080	Network instances restriction is enabled
wlRegRestrictionInstallDate 0x0100	Install before specific date is enabled
wlRegRestrictionCreationDate 0x0200	License has its own creation date embedded
wlRegRestrictionEmbedUserInfo 0x0400	License information is included inside SmarKey license (for Dynamic SmartKeys only)

1.7.2.17 WLRegGetLicenseType

The **WLRegGetLicenseType** function retrieves a bitmask with the location (File or Registry) and type (Normal Key, Static SmartKey, Dynamic SmartKey) of the current license.

Show C/C++ function definition

```
int WLRegGetLicenseType(void);
```

Show Delphi function definition

```
function WLRegGetLicenseType():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegGetLicenseType Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegGetLicenseType();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegGetLicenseType() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is a bit mask with the the location and type of the current key:

- Bit 0: Location of license:
  0: License is located in a file.
  1: License is located in the Registry.

- Bits 1-2: Type of license:
  01: Normal license.
  10: Static SmartKey.
  11: Dynamic SmartKey.

1.7.2.18 WLRegGetStatus

The **WLRegGetStatus** function retrieves information about the licensing status of the current application.

Show C/C++ function definition

```
int WLRegGetStatus(
     int* pExtendedInfo
);
```

Show Delphi function definition

```
function WLRegGetStatus(
     var pExtendedInfo:Integer
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegGetStatus Lib "WinLicenseSDK.dll" (
     pExtendedInfo As Any
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegGetStatus(ref int ExtendedInfo);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegGetStatus(ByRef ExtendedInfo As Integer) As Integer
End Function
```

Parameters

pExtendedInfo

[out] Pointer to a variable that will receive extended information about the application status when "*wlLicenseExpired (5)*" is returned. When a different value from wlLicenseExpired is returned by WLRegGetStatus, this parameter contains zero.

The returned extended information can be:

- *wlLicenseDaysExpired (1)* when a license is expired on days.
- *wlLicenseExecExpired (2)* when a license is expired on executions.
- *wlLicenseDateExpired (3)* when a license is expired on date.
- *wlLicenseGlobalExpired (4)* when the global time in a license has expired.
- *wlLicenseRuntimeExpired (5)* when license runtime has expired.

Return Values

The return value is the licensing status in the current application. The possible values are the followings:

- wlIsTrial *(0)* when an application is running in trial mode (not registered).
- wlIsRegistered *(1)* when an application is registered with a valid license key.
- wlInvalidLicense *(2)* when a license key is invalid.
- wlInvalidHardwareLicense *(3)* when the machine ID, inside a license key, is invalid for current machine.
- wlNoMoreHwdChanges *(4)* when no more machine ID changes are allowed in the current license key.
- wlLicenseExpired *(5)* when a license key has expired. Check *pExtendedInfo* to retrieve extra information.
- wlInvalidCountryLicense *(6)* when a license is locked to different country.
- wlLicenseStolen *(7)* when a license key is stolen.
- wlWrongLicenseExp *(8)* when a license key is a permanent key and only licenses that expire are allowed in an application.
- wlWrongLicenseHardware *(9)* when the current license key does not have machine ID information (and machine ID is required in an application).

- wlNoMoreInstancesAllowed *(12)* when the protected application has reached the maximum number of instances in a network.
- wlNetworkNoServerRunning *(13)* when a network license is locked to a server and the server is not running the protected application.
- wlInstallLicenseDateExpired *(14)* when a license has been installed after a specific installation date.
- wlLicenseDisabledInstance *(15)* when a license has been disabled for a specific application (When calling the function WLRegDisableKeyInCurrentInstance 148).
- wlNetworkCannotStartServer (17) when the server instance cannot be started in a network license. Check firewall and IP address.

See Also

WLTrialGetStatus 126

1.7.2.19 WLRegGlobalTimeLeft

The **WLRegGlobalTimeLeft** function retrieves the number of minutes left in the current license key.

Show C/C++ function definition

```
int WLRegGlobalTimeLeft(void);
```

Show Delphi function definition

```
function WLRegGlobalTimeLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegGlobalTimeLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegGlobalTimeLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegGlobalTimeLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of minutes left in the current license key.

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

1.7.2.20 WLRegLicenseCreationDate

The **WLRegLicenseCreationDate** function retrieves the date when the license was created.

Show C/C++ function definition

```
bool WLRegLicenseCreationDate(
     SYSTEMTIME* pCreationDate
);
```

Show Delphi function definition

```
function WLRegLicenseCreationDate(
     var pCreationDate:SYSTEMTIME
):Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegLicenseCreationDate Lib "WinLicenseSDK.dll" (
     pCreationDate As Any
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegLicenseCreationDate(SystemTime ExpDate);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegLicenseCreationDate(ExpDate As SystemTime) As
     Boolean
End Function
```

Parameters

pCreationDate

[out] Pointer to a SYSTEMTIME structure that receives the license creation date.

Return Values

If the function succeeds the return value is True (1).

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

1.7.2.21 WLRegLicenseName

The **WLRegLicenseName** function retrieves the license file and registry names for the expected registration license.

Show C/C++ function definition

```
void WLRegLicenseName(
     char* pFileKeyName,
     char* pRegKeyName,
     char* pRegKeyValueName
);
```

Show Delphi function definition

```
procedure WLRegLicenseName(
     pFileKeyName: PAnsiChar;
     pRegKeyName: PAnsiChar;
     pRegKeyValueName: PAnsiChar
); stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegLicenseName Lib "WinLicenseSDK.dll" (
     ByVal pFileKeyName As String,
     ByVal pRegKeyName As String,
     ByVal pRegKeyValueName As String
) As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegLicenseName(StringBuilder FileKeyName,
     StringBuilder RegKeyName, StringBuilder RegKeyValueName);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegLicenseName(ByRef FileKeyName As String, ByRef
     RegKeyName As String, ByRef RegKeyValueName As String) As Boolean
End Function
```

Parameters

pFileKeyName
: [out] Pointer to a buffer that will receive the name of the file key expected.

pRegKeyName
: [out] Pointer to a buffer that will receive the name of the registry key expected.

RegKeyValueName
: [out] Pointer to a buffer that will receive the name of the registry value key expected.

Return Values

The function has no return values.

1.7.2.22 WLRegLockedCountry

The **WLRegLockedCountry** function retrieves the country code in the current license key. The application will be locked to that country when registered.

Show C/C++ function definition

```
int WLRegLockedCountry(void);
```

Show Delphi function definition

```
function WLRegLockedCountry():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegLockedCountry Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegLockedCountry();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegLockedCountry() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

If the function succeeds the return value is the country code in the current license key. To see a list of the possible country codes, check the country codes list.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

1.7.2.23 WLRegNetInstancesGet

The **WLRegNetInstancesGet** function retrieves the current number of instances running in a network.

Show C/C++ function definition

```
int WLRegNetInstancesGet(void);
```

Show Delphi function definition

```
function WLRegNetInstancesGet():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegNetInstancesGet Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegNetInstancesGet();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegNetInstancesGet() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of of instances running in a network.

Restrictions

If the application is registered with a license with no network instances limit, the return value is zero.

If the application is registered with a license with network instances limit and a firewall is blocking the network communication for the protected application, the return value is zero.

1.7.2.24 WLRegNetInstancesGetClientsIp

The **WLRegNetInstancesGetClientsIp** function retrieves the IP addresses of all running clients.

Show C/C++ function definition

```
int WLRegNetInstancesGetClientsIp(
     WL_IP_ADDRESS* pBufferIPs,
     int            MaxIPsToRetrieve
);
```

Show Delphi function definition

```
function WLRegNetInstancesGetClientsIp(
     pBufferIPs: array of WL_IP_ADDRESS;
     MaxIPsToRetrieve:Integer;
):Boolean; stdcall;
```

Parameters

pBufferIPs

[out] Pointer to buffer that will receive the array of WL_IP_ADDRESSES

MaxIPsToRetrieve

[in] Maximum number of WL_IP_ADDRESSES to receive in "pBufferIPs". The size of pBufferIPs should be big enough to hold the maximum number to retrieve

Return Values

The return value is the number of WL_IP_ADDRESSES that have been retrieved.

1.7.2.25 WLRegNetInstancesMax

The **WLRegNetInstancesMax** function retrieves the maximum number of instances allowed in the current license.

Show C/C++ function definition

```
int WLRegNetInstancesMax(void);
```

Show Delphi function definition

```
function WLRegNetInstancesMax():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegNetInstancesMax Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegNetInstancesMax();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegNetInstancesMax() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the maximum number of of instances allowed in the current license.

If the license has no network instances limit, the return value is zero.

1.7.2.26 WLRegNormalKeyCheck

The **WLRegNormalKeyCheck** validates a text key. This function should be called before installing a text key into the system.

From WinLicense 2.0, if you call this function the registration is performed on the fly and the application will go into registered mode if the key is correct. Notice that if the key contains expiration information, the application needs to be restarted (WLRestartApplication 258) in order to start the expiration process. If no expiration is inserted in the license, then the registration process is fully performed without having to restart the application.

Show C/C++ function definition

```
bool WLRegNormalKeyCheck(
     const char* pTextKey);
```

Show Delphi function definition

```
function WLRegNormalKeyCheck(
     pTextKey:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegNormalKeyCheck Lib "WinLicenseSDK.dll" (
     ByVal pTextKey As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegNormalKeyCheck(string TextKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegNormalKeyCheck(TextKey As String) As Boolean
End Function
```

Parameters

pTextKey

[in] Pointer to a null-terminated string that specifies the text key to validate.

Return Values

If the text key is valid, the return value is *True*.

If the text key is invalid, the return value is *False*.

See Also

WLRegNormalKeyInstallToRegistry 174, WLRegNormalKeyInstallToFile 172, WLGenLicenseTextKey 211

1.7.2.27 WLRegNormalKeyCheckW

The **WLRegNormalKeyCheckW** validates a text key (given as UNICODE). This function should be called before installing a text key into the system.

From WinLicense 2.0, if you call this function the registration is performed on the fly and the application will go into registered mode if the key is correct. Notice that if the key contains expiration information, the application needs to be restarted (WLRestartApplication 258) in order to start the expiration process. If no expiration is inserted in the license, then the registration process is fully performed without having to restart the application.

Show C/C++ function definition

```
bool WLRegNormalKeyCheckW(
     const wchar_t* pTextKey);
```

Show Delphi function definition

```
function WLRegNormalKeyCheckW(
     pTextKey:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLRegNormalKeyCheckW(string TextKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegNormalKeyCheckW(TextKey As String) As Boolean
End Function
```

Parameters

pTextKey
[in] Pointer to a null-terminated string that specifies the text key to validate.

Return Values

If the text key is valid, the return value is *True*.

If the text key is invalid, the return value is *False*.

See Also

WLRegNormalKeyInstallToRegistryW 175, WLRegNormalKeyInstallToFileW 173, WLGenLicenseTextKey 211

1.7.2.28 WLRegNormalKeyInstallToFile

The **WLRegNormalKeyInstallToFile** installs a license text key into a file. When this function is called, a license file (selected in the Winlicense user interface) will be created. This license file will be able to register the application the next time that is started.

Show C/C++ function definition

```
bool WLRegNormalKeyInstallToFile(
     const char* pTextKey);
```

Show Delphi function definition

```
function WLRegNormalKeyInstallToFile(
     pTextKey:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegNormalKeyInstallToFile Lib "WinLicenseSDK.dll" (
     ByVal pTextKey As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegNormalKeyInstallToFile(string TextKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegNormalKeyInstallToFile(TextKey As String) As
     Boolean
End Function
```

Parameters

pTextKey

[in] Pointer to a null-terminated string that specifies the text key to install into a license file.

The name of the license file can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegNormalKeyInstallToRegistry 174, **WLRegNormalKeyCheck** 169, **WLGenLicenseTextKey** 211

1.7.2.29 WLRegNormalKeyInstallToFileW

The **WLRegNormalKeyInstallToFileW** installs a license text key into a file. When this function is called, a license file (selected in the Winlicense user interface) will be created. This license file will be able to register the application the next time that is started.

Show C/C++ function definition

```
bool WLRegNormalKeyInstallToFileW(
     const wchar_t* pTextKey);
```

Show Delphi function definition

```
function WLRegNormalKeyInstallToFile(
     pTextKey:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLRegNormalKeyInstallToFileW(string TextKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegNormalKeyInstallToFileW(TextKey As String) As
     Boolean
End Function
```

Parameters

pTextKey

[in] Pointer to a null-terminated string that specifies the text key to install into a license file.

The name of the license file can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegNormalKeyInstallToRegistryW 175, WLRegNormalKeyCheckW 171, WLGenLicenseTextKey 211

1.7.2.30 WLRegNormalKeyInstallToRegistry

The **WLRegNormalKeyInstallToRegistry** installs a license text key into a Windows registry key. When this function is called, a Windows registry key (selected in the Winlicense user interface) will be created. This registry key will be able to register the application the next time that is started.

Show C/C++ function definition

```
bool WLRegNormalKeyInstallToRegistry(
     const char* pTextKey);
```

Show Delphi function definition

```
function WLRegNormalKeyInstallToRegistry(
     pTextKey:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegNormalKeyInstallToRegistry Lib "WinLicenseSDK.dll" (
     ByVal pTextKey As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegNormalKeyInstallToRegistry(string TextKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegNormalKeyInstallToRegistry(TextKey As String) As
     Boolean
End Function
```

Parameters

pTextKey

[in] Pointer to a null-terminated string that specifies the text key to install into the Windows registry.

The name of the Windows registry key can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegNormalKeyInstallToFile 172, WLRegNormalKeyCheck 169, WLGenLicenseTextKey 211

1.7.2.31 WLRegNormalKeyInstallToRegistryW

The **WLRegNormalKeyInstallToRegistryW** installs a license text key into a Windows registry key. When this function is called, a Windows registry key (selected in the Winlicense user interface) will be created. This registry key will be able to register the application the next time that is started.

Show C/C++ function definition

```
bool WLRegNormalKeyInstallToRegistryW(
     const wchar_t* pTextKey);
```

Show Delphi function definition

```
function WLRegNormalKeyInstallToRegistryW(
     pTextKey:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLRegNormalKeyInstallToRegistryW(string TextKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegNormalKeyInstallToRegistryW(TextKey As String) As
     Boolean
```

```
End Function
```

Parameters

pTextKey

[in] Pointer to a null-terminated string that specifies the text key to install into the Windows registry.

The name of the Windows registry key can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegNormalKeyInstallToFileW 173, WLRegNormalKeyCheckW 171, WLGenLicenseTextKey 211

1.7.2.32 WLRegRemoveCurrentKey

The **WLRegRemoveCurrentKey** function removes the current license key from the system. This function deletes the current license from the system (file or registry license), making an application not to be registered the next time that is executed. Notice that this function will not block the current license.

Show C/C++ function definition

```
bool WLRegRemoveCurrentKey(void);
```

Show Delphi function definition

```
function WLRegRemoveCurrentKey():Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegRemoveCurrentKey Lib "WinLicenseSDK.dll" () As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
```

```
    public static extern bool WLRegRemoveCurrentKey();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegRemoveCurrentKey() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegDisableCurrentKey 147

1.7.2.33 WLRegRuntimeLeft

The **WLRegRuntimeLeft** function retrieves the runtime left that a registered application can be running in memory.

Show C/C++ function definition

```
int WLRegRuntimeLeft(void);
```

Show Delphi function definition

```
function WLRegRuntimeLeft():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegRuntimeLeft Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegRuntimeLeft();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
```

```
    Public Shared Function WLRegRuntimeLeft() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the runtime left in minutes.

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

1.7.2.34 WLRegSmartKeyCheck

The **WLRegSmartKeyCheck** validates a SmartActivate® key. This function should be called before installing a SmartActivate® key into the system.

From WinLicense 2.0, if you call this function the registration is performed on the fly and the application will go into registered mode if the key is correct. Notice that if the key contains expiration information, the application needs to be restarted (WLRestartApplication 258) in order to start the expiration process. If no expiration is inserted in the license, then the registration process is fully performed without having to restart the application.

Show C/C++ function definition

```
bool WLRegSmartKeyCheck(
     const char* pUserName,
     const char* pOrganization,
     const char* pCustomData,
     const char* pSmartKey
     );
```

Show Delphi function definition

```
function WLRegSmartKeyCheck(
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pSmartKey:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegSmartKeyCheck Lib "WinLicenseSDK.dll" (
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pSmartKey As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyCheck(string UserName, string Company,
     string CustomData, string SmartKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyCheck(UserName As String, Company As
     String, CustomData As String, SmartKey As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to validate.

If this parameter is zero, it means that the SmartActivate® key does not have user name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to validate.

If this parameter is zero, it means that the SmartActivate® key does not have organization information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to validate.

If this parameter is zero, it means that the SmartActivate® key does not have custom information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to validate.

Return Values

If the SmartActivate® key is valid, the return value is *True*.

If the SmartActivate® key is invalid, the return value is *False*.

See Also

WLRegSmartKeyInstallToRegistry 189, WLRegSmartKeyInstallToFile 182, WLGenLicenseSmartKey 221

1.7.2.35 WLRegSmartKeyCheckW

The **WLRegSmartKeyCheckW** validates a SmartActivate® key. This function should be called before installing a SmartActivate® key into the system.

From WinLicense 2.0, if you call this function the registration is performed on the fly and the application will go into registered mode if the key is correct. Notice that if the key contains expiration information, the application needs to be restarted (WLRestartApplication 258) in order to start the expiration process. If no expiration is inserted in the license, then the registration process is fully performed without having to restart the application.

This function should be used when a license has been generated with UNICODE functions.

Show C/C++ function definition

```
bool WLRegSmartKeyCheckW(
     const wchar_t* pUserName,
     const wchar_t* pOrganization,
     const wchar_t* pCustomData,
     const wchar_t* pSmartKey
     );
```

Show Delphi function definition

```
function WLRegSmartKeyCheckW(
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pSmartKey:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyCheckW(string UserName, string Company,
     string CustomData, string SmartKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyCheckW(UserName As String, Company As
     String, CustomData As String, SmartKey As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to validate.

If this parameter is zero, it means that the SmartActivate® key does not have user name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to validate.

If this parameter is zero, it means that the SmartActivate® key does not have organization information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to validate.

If this parameter is zero, it means that the SmartActivate® key does not have custom information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to validate.

Return Values

If the SmartActivate® key is valid, the return value is *True*.

If the SmartActivate® key is invalid, the return value is *False*.

See Also

WLRegSmartKeyInstallToRegistryW 191, **WLRegSmartKeyInstallToFileW** 183, **WLGenLicenseSmartKey** 221

1.7.2.36 WLRegSmartKeyInstallToFile

The **WLRegSmartKeyInstallToFile** installs a SmartActivate® key into a file. When this function is called, a license file (selected in the Winlicense user interface) will be created. This license file will be able to register the application the next time that it is started.

Show C/C++ function definition

```
bool WLRegSmartKeyInstallToFile(
     const char* pUserName,
     const char* pOrganization,
     const char* pCustomData,
     const char* pSmartKey
     );
```

Show Delphi function definition

```
function WLRegSmartKeyInstallToFile(
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pSmartKey:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegSmartKeyInstallToFile Lib "WinLicenseSDK.dll" (
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pSmartKey As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyInstallToFile(string UserName, string
     Company, string CustomData, string SmartKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyInstallToFile(UserName As String, Company
     As String, CustomData As String, SmartKey As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have custom data information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to install into a license file.

The name of the license file can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegSmartKeyInstallToRegistry 189, WLRegSmartKeyCheck 178, WLGenLicenseSmartKey 221

1.7.2.37 WLRegSmartKeyInstallToFileW

The **WLRegSmartKeyInstallToFileW** installs a SmartActivate® key into a file. When this function is called, a license file (selected in the Winlicense user interface) will be created. This license file will be able to register the application the next time that it is started.

This function should be used when a license has been generated with UNICODE functions.

Show C/C++ function definition

```
bool WLRegSmartKeyInstallToFileW(
     const wchar_t* pUserName,
     const wchar_t* pOrganization,
     const wchar_t* pCustomData,
     const wchar_t* pSmartKey
     );
```

Show Delphi function definition

```
function WLRegSmartKeyInstallToFileW(
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pSmartKey:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyInstallToFileW(string UserName, string
     Company, string CustomData, string SmartKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyInstallToFileW(UserName As String, Company
     As String, CustomData As String, SmartKey As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have custom data information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to install into a license file.

The name of the license file can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegSmartKeyInstallToRegistryW 191, **WLRegSmartKeyCheckW** 180, **WLGenLicenseSmartKey** 221

1.7.2.38 WLRegSmartKeyInstallToFileInFolder

The **WLRegSmartKeyInstallToFileInFolder** installs a SmartActivate® key into a file in a specific folder. When this function is called, a license file (selected in the Winlicense user interface) will be created. This license file will be able to register the application the next time that it is started.

WinLicense will always search for the generated file license in the same folder as the protected application. This function is rarely used but it could be used in specific situations where the license needs to be created in a different drive (like a USB drive) from where the application needs to run.

Show C/C++ function definition

```
bool WLRegSmartKeyInstallToFileInFolder(
     const char* pUserName,
     const char* pOrganization,
     const char* pCustomData,
     const char* pSmartKey,
     const char* pFolderPath
     );
```

Show Delphi function definition

```
function WLRegSmartKeyInstallToFileInFolder(
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pSmartKey:PAnsiChar;
     pFolderPath:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegSmartKeyInstallToFileInFolder Lib "WinLi-
censeSDK.dll" (
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pSmartKey As String,
     ByVal pFolderPath As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyInstallToFileInFolder(string UserName,
     string Company, string CustomData, string SmartKey, string FolderPath);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyInstallToFileInFolder(UserName As String,
     Company As String, CustomData As String, SmartKey As String,
            FolderPath As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have custom data information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to install into a license file.

The name of the license file can be set up in the Winlicense user interface.

pFolderPath

[in] Pointer to a null-terminated string that specifies the folder where the file key will be created.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegSmartKeyInstallToRegistry 189, **WLRegSmartKeyCheck** 178, **WLGenLicenseSmartKey** 221

1.7.2.39 WLRegSmartKeyInstallToFileInFolderW

The **WLRegSmartKeyInstallToFileInFolderW** installs a SmartActivate® key into a file in a specific folder. When this function is called, a license file (selected in the Winlicense user interface) will be created. This license file will be able to register the application the next time that it is started.

WinLicense will always search for the generated file license in the same folder as the protected application. This function is rarely used but it could be used in specific situations where the license needs to be created in a different drive (like a USB drive) from where the application needs to run.

Show C/C++ function definition

```
bool WLRegSmartKeyInstallToFileInFolderW(
     const wchar_t* pUserName,
     const wchar_t* pOrganization,
     const wchar_t* pCustomData,
     const wchar_t* pSmartKey,
     const wchar_t* pFolderPath
     );
```

Show Delphi function definition

```
function WLRegSmartKeyInstallToFileInFolderW(
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pSmartKey:PWideChar;
     pFolderPath:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyInstallToFileInFolderW(string UserName,
     string Company, string CustomData, string SmartKey, string FolderPath);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyInstallToFileInFolderW(UserName As String,
     Company As String, CustomData As String, SmartKey As String,
            FolderPath As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have custom data information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to install into a license file.

The name of the license file can be set up in the Winlicense user interface.

pFolderPath

[in] Pointer to a null-terminated string that specifies the folder where the file key will be created.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegSmartKeyInstallToRegistryW 191, WLRegSmartKeyCheckW 171, WLGenLicenseSmartKey 221

1.7.2.40 WLRegSmartKeyInstallToRegistry

The **WLRegSmartKeyInstallToRegistry** installs a SmartActivate® key into a Windows registry key. When this function is called, a Windows registry key (selected in the Winlicense user interface) will be created. This Windows registry key will be able to register the application the next time that is started.

Show C/C++ function definition

```
bool WLRegSmartKeyInstallToRegistry(
     const char* pUserName,
     const char* pOrganization,
     const char* pCustomData,
     const char* pSmartKey
     );
```

Show Delphi function definition

```
function WLRegSmartKeyInstallToRegistry(
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pSmartKey:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegSmartKeyInstallToRegistry Lib "WinLicenseSDK.dll" (
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pSmartKey As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyInstallToRegistry(string UserName,
     string Company, string CustomData, string SmartKey);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyInstallToRegistry(UserName As String, Com-
     pany As String, CustomData As String, SmartKey As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have custom data information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to install into a Windows registry key.

The name of the Windows registry key can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegSmartKeyInstallToFile 182, WLRegSmartKeyCheck 178, WLGenLicenseSmartKey 221

1.7.2.41 WLRegSmartKeyInstallToRegistryW

The **WLRegSmartKeyInstallToRegistryW** installs a SmartActivate® key into a Windows registry key. When this function is called, a Windows registry key (selected in the Winlicense user interface) will be created. This Windows registry key will be able to register the application the next time that is started.

This function should be used when a license has been generated with UNICODE functions.

Show C/C++ function definition

```
bool WLRegSmartKeyInstallToRegistryW(
     const wchar_t* pUserName,
     const wchar_t* pOrganization,
     const wchar_t* pCustomData,
     const wchar_t* pSmartKey
     );
```

Show Delphi function definition

```
function WLRegSmartKeyInstallToRegistryW(
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pSmartKey:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLRegSmartKeyInstallToRegistryW(string UserName,
     string Company, string CustomData, string SmartKey);

[Visual Basic]
```

```
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLRegSmartKeyInstallToRegistryW(UserName As String, Com-
     pany As String, CustomData As String, SmartKey As String) As Boolean
End Function
```

Parameters

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the SmartActivate® key to install.

If this parameter is zero, the generated SmartActivate® key will not have custom data information.

pSmartKey

[in] Pointer to a null-terminated string that specifies the SmartActivate® key to install into a Windows registry key.

The name of the Windows registry key can be set up in the Winlicense user interface.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLRegSmartKeyInstallToFileW 187, **WLRegSmartKeyCheckW** 180, **WLGenLicenseSmartKey** 221

1.7.2.42 WLRegTotalDays

The **WLRegTotalDays** function retrieves the total number of days in the current license key.

Show C/C++ function definition

```
int WLRegTotalDays(void);
```

Show Delphi function definition

```
function WLRegTotalDays():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegTotalDays Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegTotalDays();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegTotalDays() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of total days in the current license key.

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

See Also

WLRegDaysLeft 144

1.7.2.43 WLRegTotalExecutions

The **WLRegTotalExecutions** function retrieves the total number of executions in the current license key.

Show C/C++ function definition

```
int WLRegTotalExecutions(void);
```

Show Delphi function definition

```
function WLRegTotalExecutions():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRegTotalExecutions Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLRegTotalExecutions();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRegTotalExecutions() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the number of total executions in the current license key.

If the license key is a permanent key the return value is *wlPermKey (-1)*.

If a license key has not been installed, the return value is *wlNoKey (-2)*.

See Also

WLRegExecutionsLeft 149

1.7.3 Generators Functions

Software Developers can use the embedded passwords/licenses generator in the Winlicense user interface to control the management of new clients. There are some situations where a specific password/license generator needs to be used to satisfy special needs. For example, when a software developer decides to automatically handle the creation of new licenses through an authentication system. In those cases, Winlicense exports an API to create custom password/license managers.

The following functions can be used to generate custom passwords and licenses managers.

Function	Description
WLGenLicenseFileKey 195	Generates a license FILE key to register a specific application.
WLGenLicenseFileKeyEx 200	Generates a license key that will work in a specific application.
WLGenLicenseRegistryKey 203	Generates a license REGISTRY key to register a specific application.
WLGenLicenseRegistryKeyEx 207	Generates a license key that will work in a specific application.
WLGenLicenseTextKey 211	Generates a license TEXT key to register a specific application.
WLGenLicenseTextKeyEx 215	Generates a license key that will work in a specific application.
WLGenLicenseDynSmartKey 218	Generates a license key that will work in a specific application.
WLGenLicenseSmartKey 221	Generates a license SmartActivate® key to register a specific application.
WLGenPassword 225	Generates a specific password for a given user name.
WLGenTrialExtensionFileKey 226	Generates a trial extension FILE key to extend the current trial period in a specific application.
WLGenTrialExtensionRegistryKey 229	Generates a trial extension REGISTRY key to extend the current trial period in a specific application.

1.7.3.1 WLGenLicenseFileKey

The **WLGenLicenseFileKey** function generates a license key that will work in a specific application. The generated license key should be copied directly into a file, making it the license file to register a specific application.

This function is also implemented in UNICODE as **WLGenLicenseFileKeyW** (See definition below)

Show C/C++ function definition

```
int WLGenLicenseFileKey(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     int CountryId,
     int Runtime,
     int GlobalTime,
     char* pBufferOut
);

int WLGenLicenseFileKeyW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     int CountryId,
     int Runtime,
     int GlobalTime,
     char* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseFileKey(
     pLicenseHash:PChar;
     pUserName:PChar;
     pOrganization:PChar;
     pCustomData:PChar;
     pMachineID:PChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     CountryId:Integer;
     Runtime:Integer;
     GlobalTime:Integer;
     pBufferOut:PChar
 ):Integer; stdcall;

function WLGenLicenseFileKeyW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     CountryId:Integer;
     Runtime:Integer;
     GlobalTime:Integer;
     pBufferOut:PChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseFileKey Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     ByVal NumDays As Integer,
     ByVal NumExec As Integer,
     pExpirationDate As Any,
     ByVal CountryId As Integer,
     ByVal Runtime As Integer,
     ByVal GlobalTime As Integer,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseFileKey", CallingCon-
     vention = CallingConvention.StdCall )]

     public static extern int WLGenLicenseFileKey(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         int CountryId,
         int Runtime,
         int GlobalTime,
         byte[] pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseFileKeyW", CallingConvention = CallingConvention.St-
     dCall )]

     public static extern int WLGenLicenseFileKeyW(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         int CountryId,
         int Runtime,
         int GlobalTime,
         byte[] pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

NumDays

[in] Number of days to restrict the use of the generated license key.

If this parameter is zero, the generated license key will not have days restrictions.

NumExec

[in] Number of executions to restrict the use of the generated license key.

If this parameter is zero, the generated license key will not have executions restrictions.

pExpirationDate

[in] Pointer to a SYSTEMTIME struct that holds the expiration date for the generated license key.

If this parameter is zero, the generated license key will not have date expiration restrictions.

Note for C# developers

The SYSTEMTIME struct is not supported by C#. You have to define it as follows:

```
[StructLayout(LayoutKind.Sequential)]
    public class SystemTime
    {
        public short wYear;
        public short wMonth;
        public short wDayOfWeek;
        public short wDay;
        public short wHour;
        public short wMinute;
        public short wSecond;
        public short wMilliseconds;
    }
```

CountryId

[in] Country ID value to restrict the generated license key to a specific country.

If this parameter is zero, the generated license key will work in every country.

Runtime

[in] Runtime restriction in minutes for the generated license key. The registered application will only run *Runtime* minutes in every instance in memory.

If this parameter is zero, the generated license key will not have runtime restriction.

GlobalTime

[in] Global time restriction in minutes for the generated license key. The registered application cannot be executed more than *GlobalTime* minutes in general.

If this parameter is zero, the generated license key will not have global time restriction.

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the required buffer size.

This buffer should be copied directly into a file, making it the license key that will activate a specific application.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

See Also

WLGenLicenseRegistryKey 203, **WLGenLicenseTextKey** 211, **WLGenLicenseSmartKey** 221, **WLGenLicenseFileKeyEx** 200

1.7.3.2 WLGenLicenseFileKeyEx

The **WLGenLicenseFileKeyEx** function generates a license key in the same manner as WLGenLicenseFileKey 195. WinLicense 2.0 introduces this new function to cope with new license restrictions (not present in WLGenLicenseFileKey 195) and allow extending the license restrictions in future versions of WinLicense without breaking the function definition for WLGenLicenseFileKeyEx.

The WLGenLicenseInfoFileKeyEx requires as parameter a structure (sLicenseFeatures 232) which contains all the license restrictions.

Show C/C++ function definition

```
int WLGenLicenseFileKeyEx(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     char* pBufferOut
);

int WLGenLicenseFileKeyExW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     char* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseFileKeyEx(
     pLicenseHash:PAnsiChar;
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pMachineID:PAnsiChar;
     var pLicenseFeatures: sLicenseFeatures;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;

function WLGenLicenseFileKeyExW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     var pLicenseFeatures: sLicenseFeatures;
     pBufferOut:PChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseFileKeyEx Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     pLicenseFeatures As Any,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseFileKeyEx",
     CallingConvention = CallingConvention.StdCall )]

     public static extern int WLGenLicenseFileKeyEx(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         sLicenseFeatures pLicenseFeatures,
         byte[] pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseFileKeyExW", CallingConvention = CallingConvention.St-
     dCall )]

     public static extern int WLGenLicenseFileKeyExW(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         sLicenseFeatures pLicenseFeatures,
         byte[] pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

pLicenseFeatures

[in] Pointer to a sLicenseFeatures 232 structure which contains the license restrictions.

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the required buffer size.

This buffer should be copied directly into a file, making it the license key that will activate a specific application.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

See Also

WLGenLicenseFileKey 195, WLGenLicenseRegistryKey 203, WLGenLicenseTextKey 211, WLGenLicenseSmartKey 221

1.7.3.3 WLGenLicenseRegistryKey

The **WLGenLicenseRegistryKey** function generates a license key that will work in a specific application. The generated license key should be directly copied into a *.reg* file. When a user clicks on the .reg file, the license information will be inserted in a custom registry key to re-gister an specific application.

Show C/C++ function definition

```
int WLGenLicenseRegistryKey(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     int CountryId,
     int Runtime,
     int GlobalTime,
     char* pKeyName,
     char* pKeyValueName,
     char* pBufferOut
);

int WLGenLicenseRegistryKeyW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     int CountryId,
     int Runtime,
     int GlobalTime,
     wchar_t* pKeyName,
     wchar_t* pKeyValueName,
     char* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseRegistryKey(
     pLicenseHash:PAnsiChar;
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pMachineID:PAnsiChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     CountryId:Integer;
     Runtime:Integer;
     GlobalTime:Integer;
     pKeyName:PAnsiChar;
     pKeyValueName:PAnsiChar;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;

function WLGenLicenseRegistryKeyW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     CountryId:Integer;
     Runtime:Integer;
     GlobalTime:Integer;
     pKeyName:PWideChar;
     pKeyValueName:PWideChar;
     pBufferOut:PChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseRegistryKey Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     ByVal NumDays As Integer,
     ByVal NumExec As Integer,
     pExpirationDate As Any,
     ByVal CountryId As Integer,
     ByVal Runtime As Integer,
     ByVal GlobalTime As Integer,
     ByVal pKeyName As String,
     ByVal pKeyValueName As String,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseRegistryKey",
     CallingConvention = CallingConvention.StdCall )]
     public static extern int WLGenLicenseRegistryKey(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         int CountryId,
         int Runtime,
         int GlobalTime,
         string pKeyName,
         string pKeyValueName,
         byte[] pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseRegistryKeyW", CallingConvention = CallingConven-
     tion.StdCall )]

     public static extern int WLGenLicenseRegistryKeyW(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         int CountryId,
         int Runtime,
         int GlobalTime,
         string pKeyName,
         string pKeyValueName,
         byte[] pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

NumDays

[in] Number of days to restrict the use of the generated license key.

If this parameter is zero, the generated license key will not have days restrictions.

NumExec

[in] Number of executions to restrict the use of the generated license key.

If this parameter is zero, the generated license key will not have executions restrictions.

pExpirationDate

[in] Pointer to a SYSTEMTIME struct that holds the expiration date for the generated license key.

If this parameter is zero, the generated license key will not have date expiration restrictions.

Note for C# developers

The SYSTEMTIME struct is not supported by C#. You have to define it as follows:

```
[StructLayout(LayoutKind.Sequential)]
    public class SystemTime
    {
        public short wYear;
        public short wMonth;
        public short wDayOfWeek;
        public short wDay;
        public short wHour;
        public short wMinute;
        public short wSecond;
        public short wMilliseconds;
    }
```

CountryId

[in] Country ID value to restrict the generated license key to a specific country.

If this parameter is zero, the generated license key will work in every country.

Runtime

[in] Runtime restriction in minutes for the generated license key. The registered application will only run *Runtime* minutes in every instance in memory.

If this parameter is zero, the generated license key will not have a runtime restriction.

GlobalTime

[in] Global time restriction in minutes for the generated license key. The registered application cannot be executed more than *GlobalTime* minutes in general.

If this parameter is zero, the generated license key will not have a global time restriction.

pKeyName

[in] Pointer to a string that holds the registry key name where the license key is stored.

pKeyValueName

[in] Pointer to a string that holds the registry key value name where the license key is stored.

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the required buffer size.

This buffer should be copied directly into a *.reg* file, making it the license key that will activate when a user clicks on this *.reg* file.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

See Also

WLGenLicenseFileKey 195, **WLGenLicenseTextKey** 211, **WLGenLicenseSmartKey** 221

1.7.3.4 WLGenLicenseRegistryKeyEx

The **WLGenLicenseRegistryKeyEx** function generates a license key that will work in a specific application. The generated license key should be directly copied into a *.reg* file. When a user clicks on the .reg file, the license information will be inserted in a custom registry key to register an specific application.

Show C/C++ function definition

```
int WLGenLicenseRegistryKeyEx(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     char* pKeyName,
     char* pKeyValueName,
     char* pBufferOut
);

int WLGenLicenseRegistryKeyExW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     wchar_t* pKeyName,
     wchar_t* pKeyValueName,
     wchar_t* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseRegistryKeyEx(
     pLicenseHash:PAnsiChar;
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pMachineID:PAnsiChar;
     var pLicenseFeatures: sLicenseFeatures;
     pKeyName:PAnsiChar;
     pKeyValueName:PAnsiChar;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;

function WLGenLicenseRegistryKeyExW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     var pLicenseFeatures: sLicenseFeatures;
     pKeyName:PWideChar;
     pKeyValueName:PWideChar;
     pBufferOut:PWideChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseRegistryKeyEx Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     pLicenseFeatures As Any,
     ByVal pKeyName As String,
     ByVal pKeyValueName As String,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseRegistryKeyEx",
     CallingConvention = CallingConvention.StdCall )]

     public static extern int WLGenLicenseRegistryKeyEx(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         sLicenseFeatures pLicenseFeatures,
         string pKeyName,
         string pKeyValueName,
         byte[] pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseRegistryKeyExW", CallingConvention = CallingConven-
     tion.StdCall )]

     public static extern int WLGenLicenseRegistryKeyExW(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         sLicenseFeatures pLicenseFeatures,
         string pKeyName,
         string pKeyValueName,
         byte[] pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

pLicenseFeatures

[in] Pointer to a sLicenseFeatures 232 structure which contains the license restrictions.

pKeyName

[in] Pointer to a string that holds the registry key name where the license key is stored.

pKeyValueName

[in] Pointer to a string that holds the registry key value name where the license key is stored.

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the required buffer size.

This buffer should be copied directly into a *.reg* file, making it the license key that will activate when a user clicks on this *.reg* file.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

See Also

WLGenLicenseRegistryKey 203, WLGenLicenseFileKey 195, WLGenLicenseTextKey 211, WLGenLicenseSmartKey 221

1.7.3.5 WLGenLicenseTextKey

The **WLGenLicenseTextKey** function generates a license key that will work in a specific application. The generated text key should be passed as a parameter to the function WLRegInstallTextKeyToFile 172 or WLRegInstallTextKeyToRegistry 174 to register a specific application.

Show C/C++ function definition

```
int WLGenLicenseTextKey(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     int CountryId,
     int Runtime,
     int GlobalTime,
     char* pBufferOut
);

int WLGenLicenseTextKeyW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     int CountryId,
     int Runtime,
     int GlobalTime,
     wchar_t* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseTextKey(
     pLicenseHash:PAnsiChar;
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pMachineID:PAnsiChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     CountryId:Integer;
     Runtime:Integer;
     GlobalTime:Integer;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;

function WLGenLicenseTextKeyW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     CountryId:Integer;
     Runtime:Integer;
     GlobalTime:Integer;
     pBufferOut:PWideChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseTextKey Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     ByVal NumDays As Integer,
     ByVal NumExec As Integer,
     pExpirationDate As Any,
     ByVal CountryId As Integer,
     ByVal Runtime As Integer,
     ByVal GlobalTime As Integer,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseTextKey", CallingCon-
     vention = CallingConvention.StdCall )]

     public static extern int WLGenLicenseTextKey(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         int CountryId,
         int Runtime,
         int GlobalTime,
         StringBuilder pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseTextKeyW", CallingConvention = CallingConvention.St-
     dCall )]

     public static extern int WLGenLicenseTextKeyW(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         int CountryId,
         int Runtime,
         int GlobalTime,
         StringBuilder pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

NumDays

[in] Number of days to restrict the use of the generated license key.

If this parameter is zero, the generated license key will not have a days restrictions.

NumExec

[in] Number of executions to restrict the use of the generated license key.

If this parameter is zero, the generated license key will not have an executions restrictions.

pExpirationDate

[in] Pointer to a SYSTEMTIME struct that holds the expiration date for the generated license key.

If this parameter is zero, the generated license key will not have a date expiration restrictions.

Note for C# developers

The SYSTEMTIME struct is not supported by C#. You have to define it as follows:

```
[StructLayout(LayoutKind.Sequential)]
    public class SystemTime
    {
        public short wYear;
        public short wMonth;
        public short wDayOfWeek;
        public short wDay;
        public short wHour;
        public short wMinute;
        public short wSecond;
        public short wMilliseconds;
    }
```

CountryId

[in] Country ID value to restrict the generated license key to a specific country.

If this parameter is zero, the generated license key will work in every country.

Runtime

[in] Runtime restriction in minutes for the generated license key. The registered application will only run *Runtime* minutes in every instance in memory.

If this parameter is zero, the generated license key will not have a runtime restriction.

GlobalTime

[in] Global time restriction in minutes for the generated license key. The registered application cannot be executed more than *GlobalTime* minutes in general.

If this parameter is zero, the generated license key will not have a global time restriction.

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the required buffer size.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

See Also

WLRegNormalKeyInstallToFile 172, WLRegNormalKeyInstallToRegistry 174, WLRegNormalKeyCheck 169, WLGenLicenseRegistryKey 203, WLGenLicenseSmartKey 221, WLGenLicenseFileKey 195

1.7.3.6 WLGenLicenseTextKeyEx

The **WLGenLicenseTextKeyEx** function generates a license key that will work in a specific application. The generated text key should be passed as a parameter to the function WLRegInstallTextKeyToFile 172 or WLRegInstallTextKeyToRegistry 174 to register a specific application.

Show C/C++ function definition

```
int WLGenLicenseTextKeyEx(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     char* pBufferOut
);

int WLGenLicenseTextKeyExW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     wchar_t* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseTextKeyEx(
     pLicenseHash:PAnsiChar;
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pMachineID:PAnsiChar;
     var pLicenseFeatures: sLicenseFeatures;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;

function WLGenLicenseTextKeyExW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     var pLicenseFeatures: sLicenseFeatures;
     pBufferOut:PWideChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseTextKeyEx Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     pLicenseFeatures As Any,
     ByVal pBufferOut As String
) As Integer

Public Declare Function WLGenLicenseTextKeyExW Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     pLicenseFeatures As Any,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseTextKeyEx",
     CallingConvention = CallingConvention.StdCall )]

     public static extern int WLGenLicenseTextKeyEx(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         sLicenseFeatures pLicenseFeatures,
         StringBuilder pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseTextKeyExW", CallingConvention = CallingConvention.St-
     dCall )]

     public static extern int WLGenLicenseTextKeyExW(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         sLicenseFeatures pLicenseFeatures,
         StringBuilder pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

pLicenseFeatures

[in] Pointer to a sLicenseFeatures 232 structure which contains the license restrictions.

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the required buffer size.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

See Also

WLGenLicenseTextKey 211, **WLRegNormalKeyInstallToFile** 172, **WLRegNormalKeyInstallToRegistry** 174, **WLRegNormalKeyCheck** 169, **WLGenLicenseRegistryKey** 203, **WLGenLicenseSmartKey** 221, **WLGenLicenseFileKey** 195

1.7.3.7 WLGenLicenseDynSmartKey

The **WLGenLicenseDynSmartKey** function generates a license key that will work in a specific application. The generated license key should be copied directly into a file, making it the license file to register a specific application.

This function is also implemented in UNICODE as **WLGenLicenseDynSmartKeyW** (See definition below)

Show C/C++ function definition

```
int WLGenLicenseDynSmartKey(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     char* pBufferOut
);
```

```
int WLGenLicenseDynSmartKeyW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     sLicenseFeatures* pLicenseFeatures,
     wchar_t* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseDynSmartKey(
     pLicenseHash:PAnsiChar;
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pMachineID:PAnsiChar;
     var pLicenseFeatures: sLicenseFeatures;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;

function WLGenLicenseDynSmartKeyW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     var pLicenseFeatures: sLicenseFeatures;
     pBufferOut:PWideChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseDynSmartKey Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     pLicenseFeatures As Any,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseDynSmartKey",
     CallingConvention = CallingConvention.StdCall )]


     public static extern int WLGenLicenseDynSmartKey(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
          string pCustomData,
          string pMachineID,
          sLicenseFeatures pLicenseFeatures,
          StringBuilder pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseDynSmartKeyW", CallingConvention = CallingConven-
     tion.StdCall )]

     public static extern int WLGenLicenseDynSmartKeyW(
          string pLicenseHash,
          string pUserName,
          string pOrganization,
          string pCustomData,
          string pMachineID,
          sLicenseFeatures pLicenseFeatures,
          StringBuilder pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

pLicenseFeatures

[in] Pointer to a sLicenseFeatures 232 structure which contains the license restrictions.

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the maximum size for the generated key.

This buffer should be copied directly into a file, making it the license key that will activate a specific application.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

See Also

WLGenLicenseSmartKey 221, **WLRegSmartKeyInstallToFile** 182, **WLRegSmartKeyInstallToRegistry** 189, **WLRegSmartKeyCheck** 178, **WLGenLicenseRegistryKey** 203, **WLGenLicenseFileKey** 195

1.7.3.8 WLGenLicenseSmartKey

The **WLGenLicenseSmartKey** function generates a license key that will work in a specific application. The generated SmartActivate® key should be passed as a parameter to the function **WLRegInstallSmartKeyToFile** or **WLRegInstallSmartKeyToRegistry** to register a specific application.

Show C/C++ function definition

```
int WLGenLicenseSmartKey(
     char* pLicenseHash,
     char* pUserName,
     char* pOrganization,
     char* pCustomData,
     char* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     char* pBufferOut
);

int WLGenLicenseSmartKeyW(
     wchar_t* pLicenseHash,
     wchar_t* pUserName,
     wchar_t* pOrganization,
     wchar_t* pCustomData,
     wchar_t* pMachineID,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pExpirationDate,
     wchar_t* pBufferOut
);
```

Show Delphi function definition

```
function WLGenLicenseSmartKey(
     pLicenseHash:PAnsiChar;
     pUserName:PAnsiChar;
     pOrganization:PAnsiChar;
     pCustomData:PAnsiChar;
     pMachineID:PAnsiChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;

function WLGenLicenseSmartKeyW(
     pLicenseHash:PWideChar;
     pUserName:PWideChar;
     pOrganization:PWideChar;
     pCustomData:PWideChar;
     pMachineID:PWideChar;
     NumDays:Integer;
     NumExec:Integer;
     var pExpirationDate:SYSTEMTIME;
     pBufferOut:PWideChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenLicenseSmartKey Lib "WinLicenseSDK.dll" (
     ByVal pLicenseHash As String,
     ByVal pUserName As String,
     ByVal pOrganization As String,
     ByVal pCustomData As String,
     ByVal pMachineID As String,
     ByVal NumDays As Integer,
     ByVal NumExec As Integer,
     pExpirationDate As Any,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="WLGenLicenseSmartKey",
     CallingConvention = CallingConvention.StdCall )]


     public static extern int WLGenLicenseSmartKey(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         StringBuilder pBufferOut);
}

class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", CharSet = CharSet.Unicode,
     EntryPoint="WLGenLicenseSmartKeyW", CallingConvention = CallingConvention.St-
     dCall )]

     public static extern int WLGenLicenseSmartKeyW(
         string pLicenseHash,
         string pUserName,
         string pOrganization,
         string pCustomData,
         string pMachineID,
         int NumDays,
         int NumExec,
         SystemTime pExpirationDate,
         StringBuilder pBufferOut);
}
```

Parameters

pLicenseHash

[in] Pointer to a null-terminated string that specifies the unique License hash to generate specific keys for an application.

pUserName

[in] Pointer to a null-terminated string that specifies the registration name for the license key to generate.

If this parameter is zero, the generated key will not have registration name information.

pOrganization

[in] Pointer to a null-terminated string that specifies the organization name for the license key to generate.

If this parameter is zero, the generated key will not have organization name information.

pCustomData

[in] Pointer to a null-terminated string that specifies the custom data for the license key to generate.

If this parameter is zero, the generated key will not have custom data information.

pMachineID

[in] Pointer to a null-terminated string that specifies the machine ID to lock the license key to a specific computer.

If this parameter is zero, the generated key will work in every computer (no machine locking).

NumDays

[in] Number of days to restrict the use of the generated license key. There is an internal **limit of 255 days** to keep the SmartKey size as short as possible. Dynamic SmartKeys or other keys do not contain this restriction.

If this parameter is zero, the generated license key will not have a days restrictions.

NumExec

[in] Number of executions to restrict the use of the generated license key. There is an internal **limit of 255 executions** to keep the SmartKey size as short as possible. Dynamic SmartKeys or other keys do not contain this restriction.

If this parameter is zero, the generated license key will not have an executions restrictions.

pExpirationDate

[in] Pointer to a SYSTEMTIME struct that holds the expiration date for the generated license key.

If this parameter is zero, the generated license key will not have a date expiration restrictions.

Note for C# developers

The SYSTEMTIME struct is not supported by C#. You have to define it as follows:

```
[StructLayout(LayoutKind.Sequential)]
    public class SystemTime
    {
        public short wYear;
        public short wMonth;
        public short wDayOfWeek;
        public short wDay;
        public short wHour;
        public short wMinute;
        public short wSecond;
        public short wMilliseconds;
    }
```

pBufferOut

[out] Pointer to a buffer that will hold the generated license key. If this parameter is **NULL**, the function returns the required buffer size.

Return Values

If the function succeeds, the return value is the number of bytes in the generated license key.

If the function fails, the return value is 0.

Remarks

The "**NumDays**" and "**NumExec**" parameter has an internal limit of 255 to keep the SmartKey size as short as possible. Dynamic SmartKeys 218 or other type of keys do not contain this restriction.

This function is also implemented as UNICODE: *WLGenLicenseSmartKeyW*

See Also

WLRegSmartKeyInstallToFile 182, **WLRegSmartKeyInstallToRegistry** 189, **WLRegSmartKeyCheck** 178, **WLGenLicenseRegistryKey** 203, **WLGenLicenseFileKey** 195

1.7.3.9 WLGenPassword

The **WLGenPassword** function generates a specific password for a given user name.

Show C/C++ function definition

```
int WLGenPassword(
     char* pPassHash,
     char* pUserName,
     char* pBufferOut
);
```

Show Delphi function definition

```
function WLGenPassword(
     pPassHash:PAnsiChar;
     pUserName:PAnsiChar;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenPassword  Lib "WinLicenseSDK.dll"(
     ByVal pPassHash As String,
     ByVal pUserName As String,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="GenerateLicenseFileKey",
     CallingConvention = CallingConvention.StdCall )]

     public static extern int WLGenPassword(
         string pPassHash,
         string pUserName,
         byte[] pBufferOut);
}
```

Parameters

pPassHash
[in] Pointer to a null-terminated string that specifies the unique Password hash to generate specific passwords for an application.

pUserName
[in] Pointer to a null-terminated string that specifies the user name for whom the password is generated.

pBufferOut
[out] Pointer to a buffer that will hold the generated password for the specified user.

Return Values

If the function succeeds, the return value is the number of bytes in the generated password.

If the function fails, the return value is 0.

See Also

WLValidatePassword

1.7.3.10 WLGenTrialExtensionFileKey

The **WLGenTrialExtensionFileKey** function generates a trial extension key to extend the current trial period in a specific application. The generated trial extension key should be copied directly into a file, making it the trial extension file to extend the trial period in a specific application.

Show C/C++ function definition

```
int WLGenTrialExtensionFileKey(
     char* pTrialHash,
     int Level,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pNewDate,
     int GlobalTime,
     int Runtime,
     char* pBufferOut
);
```

Show Delphi function definition

```
function WLGenTrialExtensionFileKey(
     pTrialHash:PAnsiChar;
     Level:Integer;
     NumDays:Integer;
     NumExec:Integer;
     var pNewDate:SYSTEMTIME;
     GlobalTime:Integer;
     Runtime:Integer;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenTrialExtensionFileKey  Lib "WinLicenseSDK.dll"(
     ByVal pTrialHash As String,
     ByVal Level As Integer,
     ByVal NumDays As Integer,
     ByVal NumExec As Integer,
     pNewDate As Any,
     ByVal GlobalTime As Integer,
     ByVal Runtime As Integer,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="GenerateLicenseFileKey",
     CallingConvention = CallingConvention.StdCall )]

     public static extern int WLGenTrialExtensionFileKey(
         string pTrialHash,
         int Level,
         int NumDays,
         int NumExec,
         ref SystemTime pNewDate,
         int GlobalMinutes,
         int Runtime,
         byte[] pBufferOut);
}
```

Parameters

pTrialHash

[in] Pointer to a null-terminated string that specifies the unique Trial hash to generate specific trial extension keys for an application.

Level

[in] Trial extension key level.

The trial is extended by levels to allow developers a full control of the extended keys that they generate. When a user asks for an extension key, the first level should be zero. If the same user asks again for an extension key, only extension keys with levels greater than zero will work. In general, to allow a trial period extension, only extended keys with a *Level* greater than a previous one will work. This will avoid being cheated by users re-asking for a trial extension key.

NumDays

[in] Number of days to extend the trial period.

NumExec

[in] Number of executions to extend the trial period.

pNewDate

[in] Pointer to a SYSTEMTIME struct that holds the new trial expiration date.

If this parameter is zero, the generated license key will not have a date expiration restrictions.

Note for C# developers

The SYSTEMTIME struct is not supported by C#. You have to define it as follows:

```
[StructLayout(LayoutKind.Sequential)]
    public class SystemTime
    {
        public short wYear;
        public short wMonth;
        public short wDayOfWeek;
        public short wDay;
        public short wHour;
        public short wMinute;
        public short wSecond;
        public short wMilliseconds;
    }
```

GlobalTime

[in] Number of minutes to extend the Global time trial.

Runtime

[in] Number of runtime minutes to extend the trial.

pBufferOut

[out] Pointer to a buffer that will hold the generated trial extension key.

This buffer should be copied directly into a file, making it the extension key that will extend the trial in a specific application.

Return Values

If the function succeeds, the return value is the number of bytes in the generated trial extension key.

If the function fails, the return value is 0.

Remarks

Only the trial restriction fields that were included in the protection time can be extended. For example, if an application has a trial restriction of 21 days, ONLY the number of expiration days can be extended. Any attempt to extend any other trial restriction not included in the protection time will be ignored.

See Also

WLTrialGetExtensionStatus 122

1.7.3.11 WLGenTrialExtensionRegistryKey

The **WLGenTrialExtensionRegistryKey** function generates a trial extension key to extend the current trial period in a specific application. The generated extension key should be directly copied into a *.reg* file. When a user clicks on the .reg file, the trial extension information will be inserted in a custom registry key to register an specific application.

Show C/C++ function definition

```
int WLGenTrialExtensionRegistryKey(
     char* pTrialHash,
     int Level,
     int NumDays,
     int NumExec,
     SYSTEMTIME* pNewDate,
     int GlobalMinutes,
     int Runtime,
     char* pKeyName,
     char* pKeyValueName,
     char* pBufferOut
);
```

Show Delphi function definition

```
function WLGenTrialExtensionRegistryKey(
     pTrialHash:PAnsiChar;
     Level:Integer;
     NumDays:Integer;
     NumExec:Integer;
     var pNewDate:SYSTEMTIME;
     GlobalMinutes:Integer;
     Runtime:Integer;
     pKeyName: PAnsiChar;
     pKeyValueName: PAnsiChar;
     pBufferOut:PAnsiChar
 ):Integer; stdcall;
```

Show Visual Basic function definition

```
Public Declare Function WLGenTrialExtensionRegistryKey  Lib "WinLicenseSDK.dll"(
     ByVal pTrialHash As String,
     ByVal Level As Integer,
     ByVal NumDays As Integer,
     ByVal NumExec As Integer,
     pNewDate As Any,
     ByVal GlobalMinutes As Integer,
     ByVal Runtime As Integer,
     ByVal pKeyName As String,
     ByVal pKeyValueName As String,
     ByVal pBufferOut As String
) As Integer
```

Show C# (.NET) function definition

```
class WinlicenseSDK
{
     [DllImport( "WinlicenseSDK.dll", EntryPoint="GenerateLicenseFileKey",
     CallingConvention = CallingConvention.StdCall )]

     public static extern int WLGenTrialExtensionRegistryKey(
          string pTrialHash,
          int Level,
          int NumDays,
          int NumExec,
          ref SystemTime pNewDate,
          int GlobalMinutes,
          int Runtime,
          string pKeyName,
          string pKeyValueName,
          byte[] pBufferOut);
}
```

Parameters

pTrialHash
[in] Pointer to a null-terminated string that specifies the unique Trial hash to generate specific trial extension keys for an application.

Level
[in] Trial extension key level.

The trial is extended by levels to allow developers a full control of the extended keys that they generate. When a user asks for an extension key, the first level should be zero. If the same user asks again for an extension key, only extension keys with levels greater than zero will work. In general, to allow a trial period extension, only extended keys with a *Level* greater than a previous one will work. This will avoid being cheated by users re-asking for a trial extension key.

NumDays
[in] Number of days to extend the trial period.

NumExec
[in] Number of executions to extend the trial period.

pNewDate
[in] Pointer to a SYSTEMTIME struct that holds the new trial expiration date.

If this parameter is zero, the generated license key will not have date expiration restrictions.

Note for C# developers

The SYSTEMTIME struct is not supported by C#. You have to define it as follows:

```
[StructLayout(LayoutKind.Sequential)]
    public class SystemTime
    {
        public short wYear;
        public short wMonth;
        public short wDayOfWeek;
        public short wDay;
        public short wHour;
        public short wMinute;
        public short wSecond;
        public short wMilliseconds;
    }
```

GlobalTime
[in] Number of minutes to extend the Global time trial.

Runtime

[in] Number of runtime minutes to extend the trial.

pKeyName
[in] Pointer to a string that holds the registry key name where the trial extension key is stored.

pKeyValueName
[in] Pointer to a string that holds the registry key value name where the trial extension key is stored.

pBufferOut
[out] Pointer to a buffer that will hold the generated trial extension key.

This buffer should be copied directly into a *.reg* file, making it the extension key that will extend the trial when a user clicks on this *.reg* file.

Return Values

If the function succeeds, the return value is the number of bytes in the generated trial extension key.

If the function fails, the return value is 0.

Remarks

Only the trial restriction fields that were included in the protection time can be extended. For example, if an application has a trial restriction of 21 days, ONLY the number of expiration days can be extended. Any attempt to extend any other trial restriction not included in the protection time will be ignored.

See Also

WLTrialGetExtensionStatus 122

1.7.3.12 sLicenseFeatures Definition

This structure is used in the new license generator functions (with the "Ex" postfix) to give more flexibility to WinLicense to extend the restriction features in upcoming versions of WinLicense. Notice that all fields in the structure are defined as 32-bit length (except the SYSTEMTIME fields) to simplify the migration of this structure to non common compilers.

IMPORTANT: Before calling a license generator function which uses the *sLicenseFeatures* structure, you must initialize the whole structure to ZERO and assign the *sLicenseFeatures.cb* field to the size of the structure itself. If you don't initialize the structure to ZERO, it might contain spurious data from memory which could assign non wanted restrictions to your licenses.

Show C/C++ structure definition

```
typedef struct _sLicenseFeatures
{
     unsigned     cb;
     unsigned     NumDays;
     unsigned     NumExec;
     SYSTEMTIME   ExpDate;
     unsigned     CountryId;
     unsigned     Runtime;
     unsigned     GlobalMinutes;
     SYSTEMTIME   InstallDate;
     unsigned     NetInstances;
     unsigned     EmbedLicenseInfoInKey;
     unsigned     EmbedCreationDate;
} sLicenseFeatures;
```

Show Delphi structure definition

```
type
sLicenseFeatures = record

     cb: LongWord;
     NumDays: LongWord;
     NumExec: LongWord;
     ExpDate: SYSTEMTIME;
     CountryId: LongWord;
     Runtime: LongWord;
     GlobalMinutes: LongWord;
     InstallDate: SYSTEMTIME;
     NetInstances: LongWord;
     EmbedLicenseInfoInKey: LongWord;
     EmbedCreationDate: LongWord

end;
```

Show Visual Basic 6 structure definition

```
Type sLicenseFeatures

     cb As Integer;
     NumDays As Integer;
     NumExec As Integer;
     ExpDate As SYSTEMTIME;
     CountryId As Integer;
     Runtime As Integer;
     GlobalMinutes As Integer;
     InstallDate As SYSTEMTIME;
     NetInstances As Integer;
     EmbedLicenseInfoInKey As Integer;
     EmbedCreationDate As Integer;

End Type;
```

Show Visual Basic .NET structure definition

```
Structure sLicenseFeatures

     Public cb As Integer
     Public NumDays As Integer
     Public NumExec As Integer
     Public ExpDate As SYSTEMTIME
     Public CountryId As Integer
     Public Runtime As Integer
     Public GlobalMinutes As Integer
     Public InstallDate As SYSTEMTIME
     Public NetInstances As Integer
     Public EmbedLicenseInfoInKey As Integer
     Public EmbedCreationDate As Integer

End Structure;
```

Show C# structure definition

```
public class sLicenseFeatures
{
     public unsigned cb;
     public unsigned NumDays;
     public unsigned NumExec;
     public SYSTEMTIME ExpDate;
     public unsigned CountryId;
     public unsigned Runtime;
     public unsigned GlobalMinutes;
     public SYSTEMTIME InstallDate;
     public unsigned NetInstances;
     public unsigned EmbedLicenseInfoInKey;
     public unsigned EmbedCreationDate;
}
```

Fields definition

cb

Contains the size in bytes of this structutre. This field MUST always contain the size of the *sLicenseFeatures* structure to help WinLicense determine the number of features available when there are different version of the *sLicenseFeatures* structure.

NumDays

Number of days to restrict the use of the generated license key.

If this field is zero, the generated license key will not have days expiration.

NumExec

Number of executions to restrict the use of the generated license key.

If this field is zero, the generated license key will not have executions restrictions.

ExpDate

SYSTEMTIME struct that holds the expiration date for the generated license key.

If this struct is all zero, the generated license key will not have date expiration restrictions.

CountryId

Country ID value to restrict the generated license key to a specific country.

If this field is zero, the generated license key will work in every country.

Runtime

Runtime restriction in minutes for the generated license key. The registered application will only run *Runtime* minutes in every instance in memory.

If this field is zero, the generated license key will not have a runtime restriction.

GlobalMinutes

Global time restriction in minutes for the generated license key. The registered application cannot be executed more than *GlobalTime* minutes in general.

If this field is zero, the generated license key will not have a global time restriction.

InstallDate

Expiration date to install the license. The user must install the license before the specified *InstallDate*. If a user tries to install a license after *InstallDate* the license will be rejected.

If this struct is all zero, the generated license key will not have install date restrictions.

NetInstances

Maximun number of instances running inside a network.

If this field is zero, the license will not have restriction inside a network.

EmbedLicenseInfoInKey

This field is only used by Dynamic SmartActivate 96 keys. When this field is set to 1, the generated SmartActivate key will contain the user information embedded inside the SmartActivate serial number.

If this field is 0, the user information will not be embedded inside the SmartActivate serial number.

EmbedCreationDate

This field specifies if the license creation date will be embedded inside the license. The creation date of the license can be obtained in runtime via the function WLRegLicenseCreationDate 164.

If this field is 1, the creation date will be inserted inside the generated license.

If this field is 0, the creation date is not inserted inside the license. In that case, WLRegLicenseCreationDate 164 will return *false*.

1.7.4 Miscellaneous Functions

The following set of functions can be used to set/get miscellaneous information from WinLicense.

Function	Description
WLBufferCrypt 238	Encrypts a buffer with a given password.
WLBufferDecrypt 239	Decrypts a buffer with a given password.
WLCheckVirtualPC 240	Check if a protected application is running under a Virtual Machine environment.
WLGetCurrentCountry 241	Gets the current country ID.
WLGetLastError 243	Gets extended error information for specific functions
WLGetVersion 244	Gets the WinLicense version that the application was protected with.
WLGetProtectionDate 245	Gets the date when the application was protected.
WLHardwareCheckID 246	Validates a given hardware ID.
WLHardwareGetFormattedID 247	Retrieves a formatted hardware ID for the current machine.
WLHardwareGetID 248	Retrieves the current hardware ID for the current machine.
WLHardwareGetIDW 249	Retrieves the current hardware ID for the current machine in UNICODE format.
WLHardwareGetIdType	Retrieves the type of Hardware ID obtained with the function WLHardwareGetID 248
WLHardwareGetNumberUsbDrives 250	Retrieves the number of USB drives that are found on the system.
WLHardwareGetUsbIdAt 251	Retrieves the USB drive ID string at a given position.
WLHardwareGetUsbIdAtW 253	Retrieves the USB drive ID string (UNICODE) at a given position.

WLHardwareGetUs-bNameAt 254	Retrieves the USB drive name at a given position.
WLHardwareGetUs-bNameAtW 255	Retrieves the USB drive name (UNICODE) at a given position.
WLIsProtected 256	Returns if an application has been protected by WinLicense.
WLProtectCheckCodeInteg-rity 257	Checks if the code section of your protected application has been patched in runtime.
WLProtectCheckDebugger 257	Detects the presence of a debugger in memory.
WLRestartApplication 258	Closes the current application and restarts it again.
WLRestartApplicationArgs 259	Closes the current application and restarts it again with arguments.
WLSplashHide 261	Hides the custom splash.

1.7.4.1 WLBufferCrypt

The **WLBufferCrypt** function encrypts a buffer with a given password.

Show C/C++ function definition

```
void WLBufferCrypt(
     void* pBuffer,
     int   BufferLength,
     char* pPassword
);
```

Show Delphi function definition

```
procedure WLBufferCrypt(
     pBuffer:Pointer;
     BufferLength:Integer;
     pPassword:PAnsiChar
); stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern void WLBufferCrypt(IntPtr Buffer, int BufferSize, string
     password);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Sub WLBufferCrypt(Buffer As IntPtr, BufferSize As Integer, Pass-
     word As String)
End Sub
```

Parameters

pBuffer
[in] Pointer to a buffer to encrypt.

BufferLength
[in] Size of buffer to encrypt.

pPassword
[in] Pointer to a null terminated string with the encryption password.

Return Values

This function has no return values.

Remarks

The current encryption algorithm works with block multiplication factor 4 bytes. If the length of the buffer to encrypt is not factor 4 bytes, the remaining 3, 2 or 1 bytes won't be encrypted. You might want to padd first the buffer to 4 bytes multiple (adding garbage data at the end) before calling WLBufferCrypt.

See Also

WLBufferDecrypt 239

1.7.4.2 WLBufferDecrypt

The **WLBufferDecrypt** function decrypts a buffer with a given password.

Show C/C++ function definition

```
void WLBufferDecrypt(
     void* pBuffer,
     int   BufferLength,
     char* pPassword
);
```

Show Delphi function definition

```
procedure WLBufferDecrypt(
     pBuffer:Pointer;
     BufferLength:Integer;
     pPassword:PAnsiChar
); stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern void WLBufferDecrypt(IntPtr Buffer, int BufferSize, string
     password);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Sub WLBufferDecrypt(Buffer As IntPtr, BufferSize As Integer,
     Password As String)
End Sub
```

Parameters

pBuffer

[in] Pointer to a buffer to decrypt.

BufferLength

[in] Size of buffer to decrypt.

pPassword
[in] Pointer to a null terminated string with the decryption password.

Return Values

This function has no return values.

See Also

WLBufferCrypt 238

1.7.4.3 WLCheckVirtualPC

The **WLCheckVirtualPC** function allows you to check if a protected application is running under a Virtual Machine environment (VMWare, VirtualPC, VirtualBox...)

Show C/C++ function definition

```
bool WLCheckVirtualPC (void);
```

Show Delphi function definition

```
function WLCheckVirtualPC():Boolean; stdcall
```

Show Visual Basic Native function definition

```
Public Declare Function WLCheckVirtualPC  Lib "WinLicenseSDK.dll" () As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLCheckVirtualPC();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLCheckVirtualPC() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the protected application is running under a Virtual Machine environment, the return value is *True*.

If the protected application is not running under a Virtual Machine environment, the return value is *False*.

1.7.4.4 WLGetCurrentCountry

The **WLGetCurrentCountry** function returns the current country ID.

Show C/C++ function definition

```
int WLGetCurrentCountry (void);
```

Show Delphi function definition

```
function WLGetCurrentCountry ():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLGetCurrentCountry  Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLGetCurrentCountry();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLGetCurrentCountry() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

The return value is the current country ID. The following list shows all recognized countries and their IDs:

Country	ID		Country	ID
Albania	355		Liechtenstein	41
Algeria	213		Lithuania	370

Argentina	54		Luxembourg	352
Armenia	374		Macau S.A.R., PRC	853
Australia	61		Former Yugoslav Repub-lic of Macedonia	389
Austria	43		Malaysia	60
Azerbaijan	994		Maldives	960
Bahrain	973		Mexico	52
Belarus	375		Principality of Monaco	33
Belgium	32		Mongolia	976
Belize	501		Morocco	212
Bolivia	591		Netherlands	31
Brazil	55		New Zealand	64
Brunei Darussalam	673		Nicaragua	505
Bulgaria	359		Norway	47
Canada	2		Oman	968
Caribbean	1		Islamic Republic of Pakistan	92
Chile	56		Panama	507
Colombia	57		Paraguay	595
Costa Rica	506		Peru	51
Croatia	385		Republic of the Philip-pines	63
Czech Republic	420		Poland	48
Denmark	45		Portugal	351
Dominican Republic	1		People's Republic of Ch-ina	86
Ecuador	593		Puerto Rico	1
Egypt	20		Qatar	974
El Salvador	503		Romania	40
Estonia	372		Russia	7
Faeroe Islands	298		Saudi Arabia	966
Finland	358		Serbia	381
France	33		Singapore	65
Georgia	995		Slovak Republic	421
Germany	49		Slovenia	386
Greece	30		South Africa	27
Guatemala	502		Korea	82
Honduras	504		Spain	34
Hong Kong S.A.R., P.R.C.	852		Sweden	46
Hungary	36		Switzerland	41
Iceland	354		Syria	963
India	91		Taiwan	886

Indonesia	62		Tatarstan	7
Iran	981		Thailand	66
Iraq	964		Trinidad y Tobago	1
Ireland	353		Tunisia	216
Israel	972		Turkey	90
Italy	39		U.A.E.	971
Jamaica	1		Ukraine	380
Japan	81		United Kingdom	44
Jordan	962		United States	1
Kazakstan	7		Uruguay	598
Kenya	254		Uzbekistan	7
Kuwait	965		Venezuela	58
Kyrgyzstan	996		Viet Nam	84
Latvia	371		Yemen	967
Lebanon	961		Zimbabwe	263
Libya	218			

1.7.4.5 WLGetLastError

The **WLGetLastError** function returns extended error information in some validation functions. The current functions that are returning error information are: WLRegSmartKeyCheck 178, WLRegSmartKeyCheckW 180, WLRegNormalKeyCheck 169, WLRegNormalKeyCheckW 171.

Show C/C++ function definition

```
int WLGetLastError (void);
```

Show Delphi function definition

```
function WLGetLastError ():Integer; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLGetLastError  Lib "WinLicenseSDK.dll" () As Integer
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLGetLastError();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLGetLastError() As Integer
End Function
```

Parameters

This function has no parameters.

Return Values

ERROR CODE	DESCRIPTION
WL_ERROR_SUCCESS (0)	No extended error information for last called function
WL_ERROR_NOT_MATCHING_HARDWARE_ID (1)	The hardware ID in the license does not match the current machine
WL_ERROR_NOT_MATCHING_USER_INFO (2)	The user information in the entered key is incorrect
WL_ERROR_INVALID_KEY (3)	The entered key is incorrect
WL_ERROR_REQUIRED_HARDWARE_ID (4)	The entered key does not contain a hardware ID and hardware ID was forced to be included from the Registration 36 panel (option Allow only hardware dependent (locked) registrations)
WL_ERROR_REQUIRED_EXPIRATION (5)	The entered key does not contain expiration information. Only licenses with expiration information are accepted as it was set in the Registration 36 panel (option "Accept only temporary keys (that expire)")
WL_ERROR_WRONG_KEY_SIZE (6)	The entered key has a wrong size (too big or too small)

1.7.4.6 WLGetVersion

The **WLGetVersion** function gets the WinLicense version that the application was protected with.

Show C/C++ function definition

```
void WLGetVersion(
     char* pBuffer);
```

Show Delphi function definition

```
function WLGetVersion(
     pBuffer:PAnsiChar
); stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern void WLGetVersion(StringBuilder BufferVersion);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Sub WLGetVersion(ByRef BufferVersion As String)
End Sub
```

Parameters

pBuffer
[out] Pointer to a buffer that receives the version number.

Return Values

This function has no return value.

1.7.4.7 WLGetProtectionDate

The **WLGetProtectionDate** function gets the date when the application was protected.

Show C/C++ function definition

```
void WLGetProtectionDate(
     SYSTEMTIME* pProtectionDate);
```

Show Delphi function definition

```
function WLGetProtectionDate(
     var pProtectionDate:SYSTEMTIME
); stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern void WLGetProtectionDate(SystemTime ProtectionDate);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Sub WLGetProtectionDate(ProtectionDate As SystemTime)
End Sub
```

Parameters

pProtectionDate

[out] Pointer to a SYSTEMTIME structure that receives the protection date.

Return Values

This function has no return value.

1.7.4.8 WLHardwareCheckID

The **WLHardwareCheckID** validates a given hardware ID. This function is designed to know when a user reports a fake or real hardware ID. This will avoid being cheated by users that reports the wrong typing in their hardware ID.

Show C/C++ function definition

```
bool WLHardwareCheckID (
     char* pHardwareId);
```

Show Delphi function definition

```
function WLHardwareCheckID (
     pHardwareId:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLHardwareCheckID  Lib "WinLicenseSDK.dll" (
     ByVal pHardwareId As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLHardwareCheckID(string HardwareId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareCheckID(HardwareId As String) As Boolean
End Function
```

Parameters

pHardwareId
[in] Pointer to a null-terminated string that specifies the hardware ID to validate.

Return Values

If the hardware ID is valid, the return value is *True*.

If the hardware ID is invalid, the return value is *False*.

See Also

WLHardwareGetID 248, **WLHardwareGetFormattedID** 247

1.7.4.9 WLHardwareGetFormattedID

The **WLHardwareGetFormattedID** retrieves a formatted hardware ID for the current machine. This function should be used instead of **WLHardwareGetID** 248 when a developer needs a specially formatted hardware ID in his application.

Show C/C++ function definition

```
bool WLHardwareGetFormattedID (
     int NumCharsToghether,
     int Uppercase,
     char* pHardwareId);
```

Show Delphi function definition

```
function WLHardwareGetFormattedID (
     NumCharsTogether:Integer;
     Uppercase:Integer;
     pHardwareId:PAnsiChar
):Boolean; stdcall;;
```

Show Visual Basic Native function definition

```
Public Declare Function WLHardwareGetFormattedID  Lib "WinLicenseSDK.dll" (
     ByVal NumCharsToghether As Integer,
     ByVal Uppercase As Integer,
     ByVal pHardwareId As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLHardwareGetFormattedID(int NumCharsTogether, int
     IsUppercase, StringBuilder HardwareId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetFormattedID(NumCharsTogether As Integer,
     IsUppercase As Integer, ByRef HardwareId As String) As Boolean
End Function
```

Parameters

NumCharsTogether
[in] This parameter specifies the number of characters between dashes ('-').

Uppercase
[in] This parameter specifies if the hardware ID must be given in uppercase.

pHardwareId
[out] Pointer to a buffer that will receive a null-terminated string with the formatted hardware ID for the current machine.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

See Also

WLHardwareCheckID 246, WLHardwareGetID 248

1.7.4.10 WLHardwareGetID

The **WLHardwareGetID** retrieves the current hardware ID for the current machine. This function should be used to retrieve the hardware ID in a machine to create license keys that are locked to a specific machine.

Show C/C++ function definition

```
bool WLHardwareGetID (
     char* pHardwareId);
```

Show Delphi function definition

```
function WLHardwareGetID (
     pHardwareId:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLHardwareGetID  Lib "WinLicenseSDK.dll" (
     ByVal pHardwareId As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLHardwareGetID(StringBuilder HardwareId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetID(ByRef HardwareId As String) As Boolean
End Function
```

Parameters

pHardwareId

[out] Pointer to a buffer that will receive a null-terminated string with the hardware ID for the current machine.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

Remarks

The format of the hardware ID in the current implementation of WinLicense is as follows:

XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX (where '*X*' can be [*0-9*, *A-F*])

See Also

WLHardwareCheckID 246, **WLHardwareGetFormattedID** 247

1.7.4.11 WLHardwareGetIDW

The **WLHardwareGetIDW** retrieves the current hardware ID for the current machine in UNICODE format. This function should be used to retrieve the hardware ID in a machine to create license keys that are locked to a specific machine.

Show C/C++ function definition

```
bool WLHardwareGetIDW (
     wchar_t* pHardwareId);
```

Show Delphi function definition

```
function WLHardwareGetIDW (
     pHardwareId:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#][DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode, CallingConvention =
     CallingConvention.StdCall)]
    public static extern bool WLHardwareGetIDW(StringBuilder HardwareId);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode, CallingConven-
     tion:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetIDW(HardwareId As StringBuilder) As Boolean
End Function
```

Parameters

pHardwareId

[out] Pointer to a buffer that will receive a UNICODE string with the hardware ID for the current machine.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

Remarks

The format of the hardware ID in the current implementation of WinLicense is as follows:

XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX (where '*X*' can be [*0-9*, *A-F*])

See Also

WLHardwareCheckID 246, WLHardwareGetFormattedID 247

1.7.4.12 WLHardwareGetNumberUsbDrives

The **WLHardwareGetNumberUsbDrives** function retrieves the number of USB drives from the system where a USB ID can be obtained.

Show C/C++ function definition

```
int WLHardwareGetNumberUsbDrives(void)
```

Show Delphi function definition

```
function WLHardwareGetNumberUsbDrives():Integer; stdcall;;
```

Show Visual Basic Native function definition

```
Public Declare Function WLHardwareGetNumberUsbDrives  Lib "WinLicenseSDK.dll" () As
Integer
```

Show .NET function definition

```
[C#]
 [DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern int WLHardwareGetNumberUsbDrives();

[Visual Basic]
    <DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetNumberUsbDrives() As Integer
    End Function
```

Parameters

This function has no parameters.

Return Values

Returns the number of USB drives (to obtain their hardware ID) that were found on the current system.

1.7.4.13 WLHardwareGetUsbIdAt

The **WLHardwareGetUsbIdAt** function retrieves hardware ID for a specific USB drive that was found on the system. You need to first call the function WLHardwareGetNumberUsbDrives 250 to know the number of USB drives that were found on the system. After that, you have to iterate over all found USB drives calling the function WLHardwareGetUsbIdAt.

Show C/C++ function definition

```
bool WLHardwareGetUsbIdAt(
     int   Index,
     char* pHardwareId
     );
```

Show Delphi function definition

```
function WLHardwareGetUsbIdAt(
     Index:Integer;
     pHardwareId:PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLHardwareGetUsbIdAt Lib "WinLicenseSDK.dll" (
     ByVal Index As Integer,
     ByVal pHardwareId As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLHardwareGetUsbIdAt(int Index, StringBuilder Hard-
     wareId);

[Visual Basic]
    <DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetUsbIdAt(Index As Integer, ByRef HardwareId
     As String) As Boolean
    End Function
```

Parameters

Index

[in] Index of the specific USB drive to retrieve the hardware ID

pHardwareId

[out] Pointer to a buffer that will receive a null-terminated string with the hardware ID for the USB drive at the given position.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

Remarks

The format of the hardware ID in the current implementation of WinLicense is as follows:

XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX (where '*X*' can be [*0-9*, *A-F*])

1.7.4.14 WLHardwareGetUsbIdAtW

The **WLHardwareGetUsbIdAtW** function retrieves hardware ID (as UNICODE string) for a specific USB drive that was found on the system. You need to first call the function WLHardwareGetNumberUsbDrives 250 to know the number of USB drives that were found on the system. After that, you have to iterate over all found USB drives calling the function WLHardwareGetUsbIdAtW.

Show C/C++ function definition

```
bool WLHardwareGetUsbIdAtW(
     int       Index,
     wchar_t* pHardwareId
     );
```

Show Delphi function definition

```
function WLHardwareGetUsbIdAtW(
     Index:Integer;
     pHardwareId:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode,
            CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLHardwareGetUsbIdAtW(int Index, StringBuilder Hard-
     wareId);

[Visual Basic]
    <DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode,
            CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetUsbIdAtW(Index As Integer, ByRef HardwareId
     As String) As Boolean
    End Function
```

Parameters

Index

[in] Index of the specific USB drive to retrieve the hardware ID

pHardwareId

[out] Pointer to a buffer that will receive a null-terminated UNICODE string with the hardware ID for the USB drive at the given position.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

Remarks

The format of the hardware ID in the current implementation of WinLicense is as follows:

XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX-XXXX (where '*X*' can be [*0-9*, *A-F*])

1.7.4.15 WLHardwareGetUsbNameAt

The **WLHardwareGetUsbNameAt** function retrieves the drive name for a specific found USB drive on the system. You need to first call the function WLHardwareGetNumberUsbDrives 250 to know the number of USB drives that were found on the system. After that, you have to iterate over all found USB drives calling the function WLHardwareGetUsbNameAt.

Show C/C++ function definition

```
bool WLHardwareGetUsbNameAt(
     int       Index,
     char_t*   pUsbName
     );
```

Show Delphi function definition

```
function WLHardwareGetUsbNameAt(
     Index:Integer;
     pUsbName:PAnsiChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
 [DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLHardwareGetUsbNameAt(int Index, StringBuilder Hard-
     wareId);


[Visual Basic]
    <DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetUsbNameAt(Index As Integer, ByRef Hard-
     wareId As String) As Boolean
    End Function
```

Parameters

Index
: [in] Index of the specific USB drive to retrieve the drive name

pHardwareId
: [out] Pointer to a buffer that will receive a null-terminated UNICODE string with the drive name for the USB drive at the given position.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

1.7.4.16 WLHardwareGetUsbNameAtW

The **WLHardwareGetUsbNameAtW** function retrieves the drive name (as UNICODE string) for a specific found USB drive on the system. You need to first call the function WLHardwareGetNumberUsbDrives 250 to know the number of USB drives that were found on the system. After that, you have to iterate over all found USB drives calling the function WLHardwareGetUsbNameAtW.

Show C/C++ function definition

```
bool WLHardwareGetUsbNameAtW(
     int       Index,
     wchar_t* pUsbName
     );
```

Show Delphi function definition

```
function WLHardwareGetUsbNameAtW(
     Index:Integer;
     pUsbName:PWideChar
):Boolean; stdcall;
```

Show .NET function definition

```
[C#]
    [DllImport(WINLICENSE_SDK_DLL, CharSet = CharSet.Unicode,
            CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLHardwareGetUsbNameAtW(int Index, StringBuilder
     HardwareId);

[Visual Basic]
    <DllImport("WinlicenseSDK.dll", CharSet:=CharSet.Unicode,
            CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLHardwareGetUsbNameAtW(Index As Integer, ByRef Hard-
     wareId As String) As Boolean
    End Function
```

Parameters

Index

[in] Index of the specific USB drive to retrieve the drive name

pHardwareId

[out] Pointer to a buffer that will receive a null-terminated UNICODE string with the drive name for the USB drive at the given position.

Return Values

If the function succeeds, the return value is *True*.

If the function fails, the return value is *False*.

1.7.4.17 WLIsProtected

The **WLIsProtected** function returns if an application has been protected by WinLicense

Show C/C++ function definition

```
bool WLIsProtected(void);
```

Show Delphi function definition

```
function WLIsProtected():Boolean; stdcall;
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern Boolean WLIsProtected();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLIsProtected()as Boolean
End Sub
```

Parameters

This function has no parameters.

Return Values

If the application is protected, it will return *True*. If it is not protected, it will return *False*.

1.7.4.18 WLProtectCheckDebugger

The **WLProtectCheckDebugger** checks for the presence of a debugger in memory.

Show C/C++ function definition

```
bool WLProtectCheckDebugger();
```

Show Delphi function definition

```
function WLProtectCheckDebugger():Boolean;
```

Show Visual Basic Native function definition

```
Public Declare Function WLProtectCheckDebugger Lib "WinLicenseSDK.dll" () As
     Boolean
```

Parameters

This function has no parameters.

Return Values

If a debugger is present, the return value is *True*.

If a debugger is not present, the return value is *False*.

Remarks

This function can be called with the option "*Anti-Debugger Detection*" (in *Protection Options* panel) enabled or disabled.

1.7.4.19 WLProtectCheckCodeIntegrity

The **WLProtectCheckCodeIntegrity** checks if the code section of your protected application has been patched in runtime. This function is equivalent to the CHECK_CODE_INTEGRITY 86 macro.

Show C/C++ function definition

```
bool WLProtectCheckCodeIntegrity();
```

Show Delphi function definition

```
function WLProtectCheckCodeIntegrity():Boolean;
```

Show Visual Basic Native function definition

```
Public Declare Function WLProtectCheckCodeIntegrity Lib "WinLicenseSDK.dll" () As
     Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLProtectCheckCodeIntegrity();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLProtectCheckCodeIntegrity() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the protected application code is patched, the return value is *True*.

If the protected application code is not patched, the return value is *False*.

1.7.4.20 WLRestartApplication

The **WLRestartApplication** function closes the current application and restarts it again. This function should be called when a text key or SmartActivate® key has been installed (using **WLRegInstallSmartKeyToFile** 182, **WLRegInstallSmartKeyToRegistry** 189, **WLRegInstallTextKeyToFile** 172, **WLRegInstallTextKeyToRegistry** 174) and the application needs to be restarted to finish the WinLicense registration process.

From WinLicense 2.0, if you call WLRegSmartKeyCheck 178 or WLRegNormalKeyCheck 169, the registration is performed on the fly and the application will go into registered mode if the key is correct. Notice that if the SmartKey or Text Key contains expiration information, the application needs to be restarted in order to start the expiration process. If no expiration is inserted in the license, then the registration process is fully performed without having to restart the application.

Show C/C++ function definition

```
bool WLRestartApplication (void);
```

Show Delphi function definition

```
function WLRestartApplication ():Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRestartApplication  Lib "WinLicenseSDK.dll" () As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRestartApplication();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRestartApplication() As Boolean
End Function
```

Parameters

This function has no parameters.

Return Values

If the function succeeds, the application is restarted.

If the function fails, the return value is zero.

Remarks

This function has not effect in unprotected state for .NET applications.

1.7.4.21 WLRestartApplicationArgs

The **WLRestartApplicationArgs** function closes the current application and restarts it again with arguments. This function should be called when a text key or SmartActivate® key has been installed (using WLRegInstallSmartKeyToFile 182, WLRegInstallSmartKeyToRegistry 189, WLRegInstallTextKeyToFile 172, WLRegInstallTextKeyToRegistry 174) and the application needs to be restarted to finish the WinLicense registration process.

Show C/C++ function definition

```
bool WLRestartApplicationArgs (
     char* pArgs);
```

Show Delphi function definition

```
function WLRestartApplicationArgs (
     pArgs: PAnsiChar
):Boolean; stdcall;
```

Show Visual Basic Native function definition

```
Public Declare Function WLRestartApplicationArgs  Lib "WinLicenseSDK.dll" (
     ByVal pArgs As String
) As Boolean
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern bool WLRestartApplicationArgs(string Arguments);

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Function WLRestartApplicationArgs(Arguments As String) As Boolean
End Function
```

Parameters

pArgs

[in] Pointer to a null-terminated string that specifies the parameters that are going to be passed to the application when it is restarted.

Return Values

If the function succeeds, the application is restarted.

If the function fails, the return value is zero.

Remarks

This function has not effect in unprotected state for .NET applications.

1.7.4.22 WLSplashHide

The **WLSplashHide** function hides the splash screen that was displayed by WinLicense. When WinLicense detects that the function **WLSplashHide** is called from inside your application, WinLicense will not close the splash screen according to the values selected in the Advanced Options 65 panel (Splash Settings). The splash will be closed as soon as the **WLSplashHide** is called from the protected application.

NOTE: This function is not currently supported in .NET applications.

Show C/C++ function definition

```
void WLSplashHide ();
```

Show Delphi function definition

```
procedure WLSplashHide ();
```

Show .NET function definition

```
[C#]
[DllImport(WINLICENSE_SDK_DLL, CallingConvention = CallingConvention.StdCall)]
    public static extern void WLSplashHide();

[Visual Basic]
<DllImport("WinlicenseSDK.dll", CallingConvention:=CallingConvention.StdCall)>
    Public Shared Sub WLSplashHide()
End Sub
```

Parameters

This function has no parameters.

Remarks

This function has not effect in unprotected state.

1.7.5 Testing in an unprotected state

Winlicense offers a fast way to test the different APIs that an application calls before being protected, helping developers to quickly debug their programs as if they were already protected by WinLicense.

WinLicense gives the opportunity to set up the return values of all the APIs to allow a full testing of an application before protecting it. The core of this technique is a configuration file called "*WinlicenseSDK.ini*" that holds information about the licensing/trial status set up by the developer. For an example of this configuration file, open the "*WinlicenseSDK.ini*" file.

The **WinlicenseSDK.ini** file must be in the same directory as the WinLicenseSDK.dll.

1.8 FAQ

Here there is a list with the most frequently asked questions:

1.8.1 General

- I want to set up the WinLicense database in my server, do I need to create specific tables, etc. beforehand? 268
- I'm trying to use the MySQL database that I have created in my server, but I get an error saying "Cannot connect to server on host 'xxxx.yyyy.zzzz': Connection timeout" 269
- I have problems protecting my installer. What can I do? 269
- When I launch WinLicense.exe I get the following error "WinLicense database is already in use". I have no other instances of WinLicense running 269

1.8.1.1 Can you give me a simple guide about how to create Trial periods for my application and how to register my application with WinLicense?

At first glance, WinLicense might look a bit complex to use but as son as you familiarize a bit with it you can see that it's very intuitive. Here we explain the very basic way to create trial/registration schemes for your application protected with WinLicense.

1) Go to the Software panel 363 and create a new Software (selecting your input/unprotected file, output/protected file, etc)

2) From the main WinLicense panel ("Application Information") select your "Software" and click on "Save" Project, so you can later re-use the same Project to protect your application again and/or modify the protection options.

3) Press the "Protect" button and your application will be protected. If you now run your application, it will run exactly in the same way as your unprotected application.

Now you can start adding Trial/Registration features to your application:

1) Open you saved Project and go to the Trial Settings panel 32 and check the option "This application will run in trial/restricted mode"

2) Go to the Trial Restrictions for your application, for example you can check the "Executions" restriction and set 10 executions restriction.

3) Go to the Registration panel 36 and check the option "This application can be registered, with a presence of a valid key".

4) Protect your application and run it.

You can see that each time that you launch your protected application, it will appear a “shareware” message (**MSG_ID_TRIAL_NAG**) from the Customized Dialogs panel 43. Of course, you can edit that message or set the message to not being displayed.

After launching your application 10 times (according to the previous “10 executions” restriction) your application will refuse to run (displaying the message **MSG_ID_TRIAL_EXECUTIONS_EXPIRED** from the Customized Dialogs panel)

You can create a license key for your protected application from the License Manager 363 in WinLicense GUI, or creating your own license generator using the WinLicense SDK as explained here 336.

If you selected the “File Key” type of license (from the previous Registration panel 36) you just need to put the license file key in the same folder as your protected application. After that you can launch it and it will run in Registered Mode.

From here, you have the basis to create trial/registration schemes for your application protected with WinLicense. You can use different Trial/Registration options and features and also use the WinLicense SDK, so you can “talk” with WinLicense from your source code.

1.8.1.2 I want to protect several applications concurrently via the command line, because I'm creating a specific protected application for each customer. Is it possible?

Yes, you can do it with the specific command line that you use for Themida. In latest versions we have removed all intermediate files during protection, so there are no collisions when invoking Themida concurrently. Example:

```
Themida.exe /protect MyProject1.tmd
Themida.exe /protect MyProject2.tmd
...
```

1.8.1.3 I want to include relative paths in the "Input Filename" and "Output FileName" in the User Interface in Themida. How can I do that?

You can use special constants in your input/output file names, so you don't have to rely on full paths.

Please, refer to the Special Constants in Input/Output file names 363 section**.**

1.8.1.4 Is WinLicense compatible with Delphi 2009?

Yes, WinLicense is compatible with old and new Delphi versions. Please, notice that Delphi 2009 works with UNICODE strings by default, so you might want to use the UNICODE functions of the SDK.

1.8.1.5 Is WinLicense compatible with Windows 8?

Yes, latest versions of our products are compatible with latest Windows versions. In case that a new version of Windows appears and current WinLicense is not compatible with it, we will fix it as soon as possible as that's one of our main priorities.

1.8.1.6 I'm using SetupBuilder to build and protect my application from the command line. The application is protected correctly but I don't see any log in the command line.

WinLicense will display a log with all the protection steps when you protect under Windows XP or higher.

If you want to display the log from SetupBuilder, you can write a batch file to call WinLicense and issue a "#run winlicense.bat". This will open the command window and the log will be seen.

The winlicense.bat file will contain:

WinLicense.exe /protect YouProjectName

1.8.1.7 In my program I use the "JCLDebug" routines to get exception information (line, routine, etc) when an exception occurs. The problem is that once the application is protected, I get limited debug information.

For JCL exceptions, please, add the following line in the Advanced Options panel:

```
OPTION_ADVANCED_ADD_IMPORTS=[kernel32.dll,RaiseException]
```

1.8.1.8 What's the difference between Themida and WinLicense? If I buy WinLicense, can I use it without adding license control to my software?

WinLicense is in fact Themida, plus Trial/Licensing options. Every protection option included in Themida, is included in WinLicense.

You can use WinLicense without adding license control to your software. So, if you purchase WinLicense, you don't need to buy Themida.

1.8.1.9 Are there localized versions of your products to support other languages?

Unfortunately, in version 3.0 we removed the user interface localization. We might add it in a future.

1.8.1.10 I bought a WinLicense license to protect my applications. My friend needs to protect his application. Can I protect his software with WinLicense?

No, each license is private for the company/user that acquires it. You can use your WinLicense license to protect applications that are developed by you or that you own the full copyrights.

Each protected application is watermarked with the license key information to help us track about possible misuse in WinLicense license keys. In case that we find a borrowed/stolen key, we will invalidate that license key for future updates.

1.8.1.11 I have a suggestion about a new protection feature and features for your software. Will you implement it?

We are happy to receive suggestions about possible protection features that you think should included in our products. We will try to implement your suggestions as soon as possible. Please, contact us at info@oreans.com and let us know your requests.

1.8.1.12 What programming languages are supported by WinLicense?

WinLicense supports most programming languages, like C/C++, C#, Delphi, Visual Basic, PowerBasic, PureBasic, Assembly, .NET languages, etc.

1.8.1.13 If I protect my app with the demo version and it is stable, do I have a good level of confidence that the purchased version will work also?

Some protection features are disabled (or reduced) in the DEMO version, but you can make sure that if it works in the DEMO version, it will work in the Registered version.

Anyway, in case that you find an incompatibility we can help you out and let you know what the problem is (or fix it in case that there is a bug)

1.8.1.14 What is your support policy? Do you have a minimum response and/or problem-solving time? What types of support do you offer?

We know how important is support for our customers. In this type of software (software protection) we know that support is highly important for our customers, because if we fail in supporting our customers, they will fail supporting their customers!

We try to fix any reported problem the same day as reported (of course, this cannot be always done when we cannot reproduce the problem here and we require external tests to be done in the problematic client's PC). As soon as the problem is fixed, we release a new private version with the fix for the customer who reported the problem. The fix will be available for any of our customers who require the new private version or face the same problem.

1.8.1.15 How compatible is your software with various win O/S? (e.g. Vista in different languages, 2003 server, 64bit etc.)

Our software has been tested against all Windows versions. Notice that version 3.x does not support Windows 95/98.

1.8.1.16 I don't use Windows Vista right now, but depending on our customers I will need WinLicense (and also the protected application as well) to be able to run on Windows Vista. Do you also support 64-bit operating systems such as Windows 2003 x64 and Vista x64?

It's very important for us (and for our customers!) to keep their applications fully compatible under new operating systems (like Vista, and new x64 systems). Our products support from Windows XP to latest versions of Windows (in both 32 and 64 bits).

1.8.1.17 When using WinLicense, I see that under the "Data" subfolder, there are several big files (log files) created. Can I prevent the creation of such files?

In order to avoid the creation of those log files, please, edit your WinLicense.ini file and under the "[Database]" section, add the following line:

```
DisableLog=yes
```

1.8.1.18 I want to use the WinLicense SDK in my AutoIt application, but it does not seem to work. Can you help me?

To use the WinLicense SDK in AutoIt applications, you have the add the following option in the Advanced Options 67 panel:

```
OPTION_ADVANCED_EXPLICIT_WINLICENSE_SDK=YES
```

The following code returns the Hardware ID in a compiled AutoIt script:

```
$parameter, "ptr",DllStructGetPtr($struct), "int", 1024)
$ret = DllCall("lib\WinlicenseSDK.dll","bool", "WLHardwareGetID","ptr",
DllStructGetPtr($struct))
$return = DllStructGetData($struct,1)

MsgBox(0,"WLHardwareGetId","Machine ID: " & $return)
```

1.8.1.19 I read that WinLicense uses MySQL for the database. Do I need to install MySQL?

No, WinLicense uses internal components that handle the MySQL database (for embedded and server database).

1.8.1.20 I want to set up the WinLicense database in my server, do I need to create specific tables, etc. beforehand?

If you want to work with a database located in an external server, basically you just need to create a MySQL database (password protected or not) in your server and just pass the database address and login details to WinLicense.

Go to the License Manager 363 in WinLicense and select the top menu Database. Now click on "Select" and select "Server Database". After that, WinLicense will restart asking for the database login details.

1.8.1.21 I'm trying to use the MySQL database that I have created in my server, but I get an error saying "Cannot connect to server on host 'xxxx.yyyy.zzzz': Connection timeout"

It looks that you have the MySQL port closed in your server firewall.

1) Open the MySQL port in your server

2) Go to the "Remote MySQL" (in your server cpanel) and add there the IP address of the computer that you use to access to the database from WinLicense

1.8.1.22 I have problems protecting my installer. What can I do?

Installers are very special applications. Basically an installer compress all files and create a single file (like a ZIP file). When the installer is launched, it decompress all files in runtime (like a ZIP decompressor). Before the files are decompressed, the installer normally checks for the integrity (CRC) of the whole file, to know if it has been manipulated and it will fail to run if the file has been modified.

When an installer file is protected, the protection modifies the original installer application, so it will fail on the CRC check performed by the installer runtime code.

It does not make much sense to protect an installer, as the cracker will focus on the main extracted (installed) application to crack/attack it, not the installer itself. Also, as the main file is already compressed inside the installer file, the protection does not have the chance to apply protection on your main (installed) application.

The normal procedure would be the following:

1) Compile your application

2) Protect it with our protection

3) Create the final installer (including the protected application and all required files for the installer)

1.8.1.23 When I launch WinLicense.exe I get the following error "WinLicense database is already in use". I have no other instances of WinLicense running

In very specific machines, we have seen some problems creating the initial files for the database. We have been unable to reproduce the problem yet. If that error appears, please decompress the following file in the same folder as WinLicense.exe:

https://www.oreans.com/Files/wl_default_data_folder.rar

1.8.2 Protection Options

1.8.2.1 Can I use WinLicense from a computer with no internet connection or better under a VirtualBox/VMWare environment? I was wondering if internet is required for WinLicense to work.

WinLicense does not require internet access for the protection process. You can use it on a PC with no internet connection or under VirtualBox/VMWare.

1.8.2.2 Do I need to ship SecureEngine.dll with my protected application?

The linking with SecureEngineSDK DLL is removed at protection stage. Your protected application does not require that DLL to run, so you don't need to ship it with your application.

1.8.2.3 I use Themida with Visual Studio in the custom build steps but no output (build information) is generated at all. What's the problem?

Please, pass the /shareconsole parameter when you call WinLicense via the command line. Example:

```
WinLicense.exe /protect YourProject.tmd /shareconsole
```

1.8.2.4 I'm evaluating WinLicense DEMO to protect my Windows service, but after protecting it my service does not start at all. What should I do?

Please, notice that the DEMO version displays a splash screen before the protected application starts. For Windows services, that splash screen cannot be displayed, so the protected service cannot start. You can send us your unprotected windows service and we are happy to protect it for your with our latest version, so you can check it in your computers.

1.8.2.5 How can I avoid the command line output displayed by WinLicense when protecting my project via the command line protection?

You just need to add the "/q" parameter when you call WinLicense from the command line. Example:

WinLicense.exe /protect MyProject /q

1.8.2.6 My application requires administrator's privileges to run on Vista. Will my protected application run with admin's rights?

Protected applications will run with the same privilege level as your unprotected application. If you want to raise to admin's rights your protected application, please, refer to the section "Add Manifest" in the Extra Options 65 panel.

1.8.2.7 My (native) protected application fails to run after being protected. What can I do?

If you are having problems with your protected native EXE/DLL, please, perform the following steps:

1) Uncheck the option "**Entry Point Virtualization**" and protect again. This option is not compatible with all applications and this option might not be suitable for your application

If the application still fails, go to step 2)

2) If you are using protection macros (like VM macros, etc), uncheck the option "**Encrypt Strings in VM macros**" (ANSI strings and UNICODE strings). If your application only works in UNICODE, do not set the ANSI string option. The same applies if your application is implemented with ANSI strings (do not check the UNICODE strings option). In any case, it's better to use the STR_ENCRYPT macro around those sensitive strings that you want to protect.

If the application still fails without the option "Encrypt Strings in VM macros", go to step 3)

3) Uncheck the option "Advanced API-Wrapping". If the problem is coming from here, let us know please, so we can add support for it.

4) If you are using XBundler, please, temporally uncheck the XBundler option, just to know if the problem is coming from there. In case that this option is the culprit of the problem, let us know please, so we can add support for it.

5) If you are using different protection macros, maybe the problem is coming from a "bad" inserted macros. First, you can uncheck all macros to know if the problem is coming from there. To do so, please, ***temporally*** add the following line in the Advanced Options panel:

```
OPTION_MACROS_DISABLE_ALL_MACROS=YES
```

If the problem is coming from your inserted macros, please, check that you have not inserted one of the known macros restrictions. Please, check themhere 287. Please, do not forget to remove the option "`OPTION_MACROS_DISABLE_ALL_MACROS`"

6) If the problem still persist, please, send us your unprotected application (or any other test compiled application) to reproduce the problem here and fix it for you. Of course, any files that you send us will be treated with total confidentiality.

1.8.2.8 My .NET protected application fails to run after being protected. What can I do?

If you are having problems with your protected .NET EXE, please, perform the following steps:

1) Go to the Advanced Options 67 panel and add the following line:

```
OPTION_ADVANCED_DONT_HOOK_ALL_MODULES=YES
```

If the problem persists, go to step 2)

2) If you are using XBundler, temporally uncheck it and see if the problem is coming from there. If it does, try removing some of your inserted files to see the one that is causing the problem. You can also send us any compiled test application and the problematic file to bundle to check it here and fix it.

3) If the problem persists, go to the Advanced Options panel and add the following line:

```
OPTION_ADVANCED_DOT_NET_RELOCATE=YES
```

4) If the problem persists, please, send us any compiled test application to reproduce your problem here. Of course, any files that you send us will be treated with total confidentiality.

1.8.2.9 My CodeJock application looses skinning after being protected. What can I do?

Please, go to the Advanced Options 67 panel and add the following line:

```
OPTION_ADVANCED_CODEJOCK_SUPPORT=YES
```

1.8.2.10 My protected application is flagged as a virus. What can I do?

We have been fighting with false positives since the beginning of our protection. Virus/malware writers usually use a software protector to protect their code and make it "invisible" to antivirus. Due to this, antivirus companies are more strict on packed files.

Also, some (not widely known) antivirus, have a very bad heuristic and they even report as virus any application with a slightly different PE header. They don't even look at the code inside the application. They also ignore any file that is digitally signed and report it as virus.

If you can afford to digitally sign your protected application, that should be the best solution to fight against false positives. Most (widely used) antivirus trust digitally signed files and they are not reported as false positive.

In any case, if your protected application is flagged as virus, please, try the following steps:

1) Include version information in your application before compiling it (company name, version, etc) as some antivirus do not like compressed applications without version information on it

2) Change the icon of your application in case that you are using a default compiler icon, as some antivirus shows false positive detections if you leave a standard icon.
We have seen some cases where just changing a single pixel in the application icon removes a false positive.

3) Add an internal "pre-loader" to your protected application. These "pre-loaders" are available for our registered customers. Add the following option in the Advanced Options 67 panel:

```
OPTION_ADVANCED_HEURISTIC_PRELOADER=PATH_TO_YOUR_PRELOADER_DLL
```

Protect your application. If you still have problems with false positives or want to try to decrease the number of false positives, go to step 4)

4) Add the following option in the Advanced Options panel:

```
OPTION_ADVANCED_HEURISTIC_PRETTY_NAMES=YES
OPTION_ADVANCED_HEURISTIC_FAKE_RESOURCES=YES
```

Protect your application. If you still have problems with false positives or want to try to decrease the number of false positives, go to step 5)

5) Add the following option in the Advanced Options panel:

```
OPTION_ADVANCED_HEURISTIC_ENTRY_FIRST_SECTION=YES
```

Unfortunately, there are not a specific set of options that works better for all applications. Some applications report less false positives when protecting using just one of the above options and if more options are added, more false positives are reported. Other applications require to set *all* the above options to have less false positives.

1.8.2.11 My MSVC application generates a crash dump file (.DMP) file when it crashes, so I can load and examine the crash dump file. When my application is protected the generated crash dump does not contain valid information

You have two options to match the debug information in the protected application.

- The easiest solution (but not general for all applications) is to add the following line in the Advanced Options panel:

  ```
  OPTION_ADVANCED_KEEP_DEBUG_INFO=YES
  ```

- The second solution is to use our application **minidump_patcher** to patch the DMP file and match it for the protected application. Please, contact us for more information

1.8.2.12 Is it possible to know from my application if the application has been unpacked?

WinLicense uses its own detections to know if your application has been partially unpacked. You can also use the macro CHECK_PROTECTION to know if your application has been unpacked.

1.8.2.13 I need to get a Vista logo. Is your protection compatible with Microsoft tests?

All our products have been tested against Application Verifier (AppVerifier.exe) with all options enabled and they are fully compatible with it.

1.8.2.14 I want to protect my .NET application with WinLicense, can I use an obfuscator before protecting with WinLicense?

Yes, you can (and it's recommended) to use a .NET obfuscator before protecting with WinLicense, so your assemblies will be obfuscated and protected.

1.8.2.15 How can I omit the output displayed by WinLicense when protecting via command line?

You just need to specify the "/q" command line parameter. Example:

```
WinLicense.exe /q /protect MyProject
```

1.8.2.16 Can you let me know which protection options affect execution speed of my application?

Default protection options should not cause an impact in the execution of your application. If your application have a very big import table (this normally happens in applications with dozens of MBs) you might notice a delay before your application is launched. Notice that the Virtual Machine settings will affect the loader speed, so your application will take more or less time to boot up. All of this is related to speed in booting up.

When your application has taken control of the CPU, it should run almost as fast as the original one (with all defaults options enabled). If you are inserting protection macros (we strongly recommend the use of VM macros) in specific functions in your application, you might notice a performance decrease if the code inside the macro is called many times per second or you have tight loops inside the VM/CodeReplace macros. If you are carefully enough and put protection macros in non critical places in your application, your protected application will have an equivalent performance as the original one.

1.8.2.17 I'm using the CHECK_CODE_INTEGRITY macro in my Delphi application but the macro always returns that my code has been modified. Any ideas?

Please, notice that some components used in Delphi/BCB, like MadExcept, makes memory patching in your code in order to hook some APIs. That patch is detected by CHECK_CODE_INTEGRITY macro, so you have to avoid using the CHECK_CODE_INTEGRITY macro if you are using one of those components that patch the code section of your application in runtime.

1.8.2.18 I see that WinLicense detects if my file on disk has been patched, but how can I detect if someone has patched my application in memory?

The macro CHECK_CODE_INTEGRITY checks the integrity of your application in memory (code section). You can use this macro and call it on specific places in your application. Notice that there is a penalty in execution when calling this macro, you should avoid calling it multiple times or in places that requires a fast processing.

You can find here 86 how to use this function.

1.8.2.19 How can I insert my own splash screen using the Plugin feature?

In the WinLicense subfolder /WinLicenseSDK/ExamplesSDK/Plugins/Examples, you can find a basic example to create a plugin.

Basically, in your `SecureEngineInitialize()` function, you display your own splash screen. When `SecureEngineFinalize()` is called, you could hide your splash screen.

1.8.2.20 Can you let me know about all available Advanced Options and what they are for?

The Advanced Options panel allows you to add very specific options that are mostly related with compatibility in specific applications.

When a customer has a compatibility problem with Themida in his application, we let him know the option that he has to include in the Advanced Options panel to fix the compatibility issue.

Please, notice that the advanced options don't offer more protection to your application, but compatibility.

We try to hide those options to our customers, because the common behavior is trying to use as many options as possible to feel more secure (when basically they will add more incompatibilities to their applications)

If you are interested in a list of the available advanced options, please, contact us.

1.8.2.21 Some of my users complaint regarding RegMon (Filemon) loaded in memory. How to proceed?

If you enable the option "Detect File/Regstry Monitors" (in the Protection Options panel), WinLicense will detect common registry/file monitor tools loaded in memory. The problem with Regmon, FileMon and Process Monitor is that the driver is loaded all the time in memory even if you close the User Interface for Regmon, Filemon, etc. So, the File system and Registry are still hooked by the monitor driver until you restart the computer. Looks that the developers of those monitor tools are not unloading the driver to avoid system crashes in case that a packet request is in the middle of processing while unloading the driver. Summing up, you customer needs to restart the PC if they have launched Regmon, Filemon, etc. before launching your protected application.

1.8.2.22 If I want to sacrifice a minimal amount of security to gain the maximum amount of application startup speed, what options should I disable in WinLicense?

You should select a fast VM in the Virtual Machine panel, that will produce a noticeable startup speed.

1.8.2.23 Can I compress my application with UPX and then protect it with WinLicense?

Please, notice that WinLicense might not be compatible with already compressed/packed applications. Some compressors/packers make their own checksums to know if the file has been modified, in that case the application cannot be protected by WinLicense (or any other compressor/protector)

Another important thing to notice is that from the protection point of view, WinLicense works much better with unprotected (not packed) files, because it has access to the original application code/data in order to perform the protection.

Another alternative is to protect with WinLicense and put a compressor on top.

1.8.2.24 Is there any issue to run my protected plugins on Xp 64 (32bit mode)?

There should not be any issues under XP 64-bit. The protected DLL should be as compatible as the unprotected DLL.

1.8.2.25 My application's main function is the scientific calculation needed high performance. Is there any performace lost when I encrypt my app. with WinLicense?

Your protected application should run almost at the same speed as the original one. As you might know, WinLicense offers the chance to include protection macros (like VM, MUTATE, etc) in your application to fully virtualize the code inside the macro, where the code is emulated and never decrypted back.

You have to avoid putting those macros in critical places in your application (like code that is executed many times per second) because the execution of virtualized code is much slower than the original code. A good place to put those macros is in serial/password checking, license checks, etc.

1.8.2.26 Does WinLicense encrypt string constants in my code?

WinLicense will encrypt them but will be decrypted when your application takes control.

If you want to specifically keep encrypted some strings, you can use the macro STR_ENCRYPT 80.

1.8.2.27 If I set the Anti-Patching option, can I digitally sign my application?

Yes, the Anti-File patching option is compatible with digital certificates. After protecting your application, you can use Signtool.exe to digitally sign your protected application.

1.8.2.28 I would like to include the same protection options and custom dialogs in all my applications. Can I apply the same settings to all my applications?

Yes, to do so just save a project file with all the protection options and your customized messages. After that you just load your project file and change the name of the file to protect.

Another alternative is that you have a common project file and use it to protect different applications from the command line. Example:

```
WinLicense.exe /protect MyGlobalProjectFile.tmd /inputfile Applica-
tion1.exe /outputfile Application1_protected.exe
WinLicense.exe /protect MyGlobalProjectFile.tmd /inputfile Applica-
tion2.exe /outputfile Application2_protected.exe
```

1.8.2.29 I want to include WinLicense in my build system. Does WinLicense support command line protection?

Yes, WinLicense supports command line protection.

Please, refer to the section Protecting through the command line 70.

1.8.2.30 Can I protect my Windows NT system service with WinLicense?

Yes, WinLicense can detect which applications are Windows NT system services, so you can protect them just like normal applications. We have found few NT services that need to have resources not encrypted and not compressed. If you have problems protecting your NT service, please, uncheck the option "Compress and Encrypt --> Resources" (in the Protection Options panel) and protect again.

1.8.2.31 When I protect my application with WinLicense, the size is increased by 500Kb or more!

WinLicense adds protection code to keep your application fully protected against cracking. The protection code that is embedded into your application has about 500 Kb in size (depending on the protection options that you select). So, if you have a 50Kb application and our compression engine decrease it to 10Kb, the final protected application will be 10Kb + 500Kb in size. That’s the reason why you see your final application with bigger size.

Suppose that you have a 4000 Kb application and the compression module compress it to 2000 Kb, the final protected application will be 2000 Kb + 500 Kb = 2500 Kb, so you can see here a decrease in the final size of your protected application.

Please, go to the "Virtual Machine" panel and select a lighter Virtual Machine (like FISH (White)) to make your protected application smaller. Depending on the selected Virtual Machine the final application size will be affected considerably. In the "Virtual Machine" panel you have some statics about the size/speed/complexity of each specific Virtual Machine.

1.8.2.32 How many KB will my application grow in size, after being protected by WinLicense?

It depends on the protection options that you include. The most noticeable options are in the Virtual Machine panel. You can see different types of Virtual Machines and an approximate size for each one.

1.8.2.33 When I use macros directly around some API calls I get errors in WinLicense saying that one of my START or END markers is missing. What's wrong?

Please, notice that macros are in fact "dummy" code that does nothing (it's just recognized by WinLicense). If you put the VM_END at the end of a procedure, the compiler might remove the VM_END code when optimizations are enabled. If you compile with optimizations disabled, you should not have any problems.

The following code might generate a "missing" macro pair:

```
YourFunction()
{
    VM_START
    // your code goes here
    return;
    VM_END
}
```

The compiler knows that after the "return" statement, no code is executed, so it won't generate code for the VM_END marker. To avoid the above problem, just put it like:

```
YourFunction()
{
    VM_START
    // your code goes here
    VM_END
    return;
}
```

Also, you might want to disable optimizations in those places where macros are used, to make sure that compiler optimizations won't remove any macro markers. For C/C++ language you can use the following directive:

```
#pragma optimize("", off)
YourFunction()
{
    VM_START
    // your code goes here
    return;
    VM_END
}
#pragma optimize("", on)
```

You should be able to use the same approach for other programming languages.

1.8.2.34 Please let us know how WinLicense influences the program performance? What would you advice us to pay attention to in order to minimize the performance losses? Will it affect the protection?

As you might know, you can use macros (like VM, MUTATE...) to protect more the sensitive code in your application. Those macros are executed much slower than the real code (but they are highly recommended to put much protection in your application). So, you have to make sure that you don’t put those macros in critical code in your application (that is executed many times per second). A good place to put those macros is in code that checks for serial, passwords or other routines that are not executed constantly.

1.8.2.35 If I use the option "Entry Point Virtualization", my DLL crashes. If I uncheck that option, will it make it easier to crack?

Please, notice that Entry Point Virtualization is not compatible with some applications (specially with some compilers). When you virtualize your Entry Point, it will be destroyed and emulated somewhere else in the protection memory. Some compilers, jump at a later point in the middle of the Entry Point code, so as it’s virtualized it will execute invalid opcodes and will produce a crash in the protected application. In that case, you have to protect your application unchecking the option “Entry Point Virtualization”.

Notice that Entry Point Virtualization has the same effect as putting a VM macro in the beginning of your main function. So, you can just use VM macros in several parts of your application and you will keep a high level of protection inside your application.

1.8.2.36 Can I protect my .NET applications with WinLicense?

Yes, .NET applications (EXE) can be protected with WinLicense. Please, notice that our .NET protection is not as strong as the native protection, due to the nature of the .NET interpreted language and also our protection macros (VM, MUTATE, etc.) are not available for .NET

Notice that .NET DLLs cannot be protected with WinLicense.

1.8.2.37 Can I protect mixed managed/unmanaged DLLs?

We have done several tests on mixed managed/unmanaged DLLs and they can be protected. If you have problems with your mixed managed/unmanaged DLL, let us know so we can work on it.

1.8.2.38 When I enable the "Advanced API-Wrapping" option my applications runs slower

The Advanced API-Wrapping options has a very small penalty in execution, but it should not be noticeable in most applications. For your case, it seems that a specific API is called massively per second and you might notice a performance decrease. We have a special tool to detect which API is called massively so you can continue using the "Advanced API-Wrapping" option and just skip that specific API from wrapping (using the Advanced Options 67 panel). Please, contact us for more information at support@oreans.com

1.8.2.39 When I protect my application with an older version, the size of the protected application is smaller. Can I keep the same size in latest version?

We try to change our protection from version to version and sometimes we add more complexity to the current internal protection, making the protected application bigger.

You can go to the "Virtual Machine" panel and select a lighter/smaller VM, like VM_FISH_WHITE. You can also change the internal compressor engine to use one with better compression ratio. Please, contact us at support@oreans.com to know how to change the internal compression engine.

1.8.2.40 When I add a JPG image in the splash option, my image is not displayed on startup

Make sure that your JPG image uses the RGB color mode. JPG images with CMYK color mode are not supported yet.

1.8.3 Macros

1.8.3.1 I have a function with a VM_START/END. Inside the START - END macro markers, I call an external function, called "Function2()". Is that external "Function2()" also virtualized?

No. In that case, you will virtualize the calling convention for Function2() but not the code inside Function2(). Example:

```
void MyMainFunction()
{
   VM_START
      // some code
      Function2(1, 2, 3)
      // some code
   VM_END
}
```

In the above example, all the code inside the START - END markers is virtualized. The calling convention (passing parameters) for Function2() is also virtualized, but not the code inside "Function2()". If you also want to virtualize the code inside Function2(), you will have to put *another* macro inside Function2(). Example:

```
void Function2(param1, param2, param3)
{
    VM_START
    // some code
    VM_END
}
```

1.8.3.2 I have put a VM macro in my "main()" function. Inside the VM_START/END markers I'm calling several functions. Are those called functions also virtualized?

Suppose the following example:

```
int main(void)
{
    VM_START

    MyFunction1();
    MyFunction2();

    return 0;

    VM_END
}
```

?
In the above example you are protecting the code inside the "main()" function but not the code inside "Function1()" and "Function2()".

If you want to protect/virtualize also the code inside "Function1()" and "Function2()" you need to put another VM macros inside "Function1()" and another VM macro inside "Function2()"

1.8.3.3 I have a few Portuguese strings in my STR_ENCRYPT macro but some of them are not recognize when I click on the STR_ENCRYPT macro in the "Protection Macros" panel. What's wrong?

WinLicense searches for printable chars to be able to determine if a pointer to "something" is a string or not. As your string might contain special Portuguese characters the internal function to determine if a specific char is printable might fail.

You can change the current locale, so WinLicense will be able to find your Portuguese string.

1) Edit your WinLicense.ini file and add under the **[General]** section add the following entry:

StrEncryptLocale = Portuguese

2) Restart WinLicense and go to the "Protection Macros" panel and click on your STR_ENCRYPT macro and make sure that your Portuguese strings can now be recognized

You can change the "StrEncryptLocale" entry for a different language, like "Russian", "Spanish", etc.

1.8.3.4 Can I use one protection macro (VM macro) inside another macro (VM macro)?

You cannot nest VM macros. The following is incorrect:

```
void MyFunction()
```

```
{
   VM_START

    // your code

    VM_START    // <-- NESTED!!!!

      // your code

    VM_END

  VM_END
}
```

There is an exception for nested macros inside VM macros. You can put STR_ENCRYPT_START/END macros inside VM macros and you can also include "single marker" macros inside VM macros (like CHECK_CODE_INTEGRITY, CHECK_PROTECTION, ...)

1.8.3.5 In the "custom_vms" folder I can see the name of the available virtual machines. Can I change the internal settings inside each ".vm" file?

We have inserted an internal CRC to avoid people changing the settings inside the .vm files. We normally deliver 3 files for each specific VM architecture. For example, the TIGER architecture contains: "Tiger white", "Tiger red" and "Tiger Black".

The "white" edition decreases the protection but increases the speed in execution.

The "red" edition is balanced between protection and speed.

The "black" edition offers more security but decreases the speed in execution.

Basically, you don't need to edit the .vm files because you normally have those files ready (for better speed or protection) in the "white, red and black" editions. If you change those settings by yourself, you could create a VM with very low protection, which is not recommended, or a VM highly complex but with very low performance (and really big in size)

Anyway, if for a special reason you need to change the VM internal settings we can let you know how to proceed.

1.8.3.6 Can I raise an exception inside a VM macro?

Yes, raising an exception is possible from VM macros.

1.8.3.7 I have seen that insertion of VM macros in try-except clauses are a bit tricky. What about try-finally clauses?

You have to insert VM macros in try-finally clauses in the same was as you do for try-except. Example:

```
try
```

```
{

    VM_START

    // your code

    VM_END

}

finally

{

    VM_START

    // your code

    VM_END

}
```

1.8.3.8 Can WinLicense macros protect switch statements and try-except clauses?

Switch-Case statements and try-except clauses cannot work with WinLicense macros in most compilers.

Compilers generate a direct jump table in the data section which directly jumps to each "case" statement. When the code is virtualized, the jump goes into a virtualized (garbage) code and it produces exception. You can use a workaround to protect your switch-case statements with WinLicense macros, like:

```
switch (var)
{

    case 0:

    VM_START

    // your code

    VM_END

    case 1:

    VM_START

    // your code

    VM_END

    ...

}
```

For try-except clauses:

```
try
{
    VM_START

     // your code

     VM_END

}

except
{

     VM_START

     // your code

     VM_END

}
```

1.8.3.9 If I protect the following code with a macro: VM_START InitializeCounters(i); VM_END. Will the InitializeCounters() function code also virtualized?

No, in your example, you are just virtualizing the code which puts the parameters in the stack and call your function. But the function "InitializeCounters" itself is not virtualized. You have to put a new VM macro inside the function "InitializeCounters" if you want to virtualize it.

1.8.3.10 I have included several VM macros inside my application. I have made sure that I have not nested any macros, but when I load my application in WinLicense user interface, I get a nested macros message. What's wrong?

If you have not inserted nested macros and you get that message, you might be facing a compiler optimizations issue. Some compilers remove the VM_END marker when they are at the end of a procedure (or after a "return" statement) because the VM markers are in fact dummy code that might be affected by compiler optimizations.

Please, make sure that you avoid putting the VM_END marker after a "return" statement or as the last instruction in a function/procedure.

You can also disable local optimizations. In C/C++, you can disable local optimizations. Example:

```
#pragma optimize("", off)

```

```
void MyFunction(void)
{
    VM_START

    // your code

    VM_END
}

#pragma optimize("", on)
```

1.8.3.11 We tried to adopt WinLicense VM macro option. But, our particular problem was performance of the game. It was very critical issue. We hope to know how we improve performance of my game.

Please, notice that when you protect an area of code in a function with a VM or CodeReplace macro, the code is converted into a highly complex and unique opcodes. Those macros are executed under a specific VM (which is included inside your protected application) that is able to understand the virtualized code. Execution of code inside the Virtual Machine is much slower than the original code.

Summing up, you have to avoid putting macros in the following cases:

1) Functions which are called many times per second

2) Functions which has tight loops ("for", "while", "do-while") which iterate multiple times

3) Critical code in your application. That is, code that must be executed as soon as possible

A real example could be the case that you want to protect a function, but it has a tight loop, so, you just have to put the loop outside of the macro.

Example:

```
void MyFunction( )
{
    VM_START

    // your code goes here

    VM_END

    for(i = 0; i < 0x100000; i++)
    {
        // your code
    }

    VM_START
    // your code goes here

    VM_END
}
```

1.8.3.12 Can VM macros protect switch statements? We are now having an issue with VM macros crashing the application.

Switch-Case statements cannot work with VM macros in most compilers.

Compilers generate a direct jump table in the data section which directly jumps to each "case" statement. When the code is virtualized, the jump goes into a virtualized (garbage) code and it produces exception. You can use a workaround to protect your switch-case statements with VM macros, like:

```
switch (var)
{
    case 0:

    VM_START

    // your code

    VM_END

    case 1:

    VM_START

    // your code

    VM_END

    ...
}
```

1.8.3.13 Where are the ENCODE/CLEAR macros that were available in version 2.x?

We removed the ENCODE/CLEAR macros as they were not strong enough compared to our virtualization macros. The ENCODE/CLEAR macros got their strength in older versions when our protection was using Ring-0 code. We later removed the Ring-0 protection and added our protection virtual machines, so our most powerful protection macros are the VM macros (VM_TIGER_WHITE, VM_FISH_RED, etc.). We encourage our customers to change their ENCODE/CLEAR macros with our newest VM macros to keep a good level of protection.

1.8.3.14 When my STR_ENCRYPT macros are processed in the last "Protection" panel, I can see "skipped" when the macro is processed. What can I do?

When the STR_ENCRYPT is "skipped" it is because there are not strings found inside the STR_ENCRYPT macro markers (START - END). If you are sure that there are strings inside the macro markers, you might be using UNICODE strings in your source code but you are using the ANSI macro "STR_ENCRYPT" instead of the UNICODE version of the macro "STR_ENCRYPTW".

You can go to the "Virtual Machine" panel and click on your STR_ENCRYPT/W macros and you will see in the lower panel the strings found inside the STR_ENCRYPT macro markers, so you can make sure that your strings are recognized by the protection.

1.8.3.15 When I compile my Delphi application in 64-bit, the compiler says that the "asm instruction is not valid" in my VM macros

In order to use the protection macros (like VM macros) when compiling a 64-bit Delphi application, you have to insert the macros via "Function names" instead of "inline assembly". Please, refer to the example in the subfolder in the "`WinLicenseSDK/ExamplesSDK\Macros\Delphi\Via Functions`"

1.8.3.16 I'm using a VM macro but it fails when using it on my Delphi application. I get an exception. Can you fix it?

You can try the following which might fix your issue:

1) Go to the Advanced Options 67 panel and add the following option:

```
OPTION_MACROS_SEH_SUPPORT=YES
```

2) Protect again

If you still have problems, please, check that you have not inserted any of the current macro restrictions. Please, check the following KB entry 287.

1.8.3.17 What does it mean that my encrypted string are not removed from the original location?

When a string is referenced from inside a STR_ENCRYPT macro or from a VM macro (when using the option Encrypt Strings in VM macros 24), the protection redirects the pointer to the string to a location inside the protection code and copy the string in that location (in an encrypted form). As the original location of the string is not referenced after being protected, the original string is patched with zero values.

In some cases, the same string is also used in a different location in your application code and due to compiler optimizations, the string is only stored once.

Show Example

```
LRESULT CALLBACK MainHandler(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
```

```
{
   switch (message)
   {
   case WM_INITDIALOG:

      STR_ENCRYPT_START

      printf("Hello World");    // "Hello World" string found and redirected the
pointer inside "printf"

      return TRUE;

      STR_ENCRYPT_END

      break;

    case WM_COMMAND:

      printf("Hello World");   // The "Hello World" string is referenced here
again, from outside a STR_ENCRYPT marker

      break;
   }
   return FALSE;
}
```

In the above example, as the "Hello World" string is also referenced from another location (outside of a STR_ENCRYPT marker) the protection cannot remove the original string "Hello World" from its original location, as it will be later referenced by a different code. In this case, there is an encrypted "Hello World" string that is referenced by the code inside the STR_ENCRYPT marker and an unencrypted "Hello World" that is used in a different code location.

In some cases, you only use a specific string one time inside the STR_ENCRYPT marker, but the string is not removed. The reasons could be:

- Your compiler is creating a pointer to the string in a different location in your application. Probably the linker is using a table of symbols that contains the pointer to all your strings
- There is a special code or data sequence in your application code that matches exactly an offset to the original string. In this case, this is not a real access to the string, it was just a coincidence that a specific code sequence matched the pointer to the string

If you are sure that there are no more access to the specific string (or to overcome the two above situations), you can go to the Advanced Options 67 panel and add the following option:

OPTION_MACROS_ENCRYPT_STRINGS_FORCE_DESTROY=YES

1.8.3.18 In VS2017 and VS2019 the debugger is not tracing correctly my function that uses a VM_START/END marker

We have contacted Microsoft about this and there is a bug on Visual Studio where the debugger incorrectly traces over a code with "inline code" (like our START - END macro markers). The bug is not present on Visual Studio 2015 and it should be fixed in a new release of Visual Studio.

Meanwhile, you can also use other type of insertion for the protection macros, you could try using the ASM module method, as described in the subfolder "\Examples\C\VC (via ASM module)"

1.8.3.19 When enabling optimizations, my VM_END marker is not found

This can be an issue with Tail Call Optimization.

To avoid the issue with the "missing END" marker due to Tail Call Optimization, please, just put an intrinsic __nop() after the END macro marker. You could also edit the definition of the END marker (or create your own wrapper for it) where you add at the end the "__nop();" instruction. Example:

```
VM_START;

MessageBoxW(0, L"Hello world!", 0, 0);

VM_END;

__nop();   // prevent TCO in this function
```

1.8.3.20 I'm using the STR_ENCRYPT but sometimes, when my string contains specific German characters the string is not recognized

We use the C function "isprint()" to determine if a char is valid. We have an external option to change the locale, so "isprint()" behaves differently (using "_wsetlocale()").

Please, edit the Winlicense.ini file and add the following line under the "[General]" tab:

```
StrEncryptLocale=de-DE
```

You can use different srtings admitted by _wsetlocale for other specific languages

1.8.4 XBundler

- Can I specify via the command line a file which contains all files that will be embedded in XBundler? 295

1.8.4.1 Can I specify via the command line a file which contains all files that will be embedded in XBundler?

The XBundler files are specified in your winlicense project file. You can directly modify the project file to set your files to bundle.

Another option is to specify a separate file in the "Advanced Options" panel that will contain the list of files to bundle. For example, you can add the following line in the "Advanced Options" panel if you want to have a separate options file (let's call it "ExternalOptions.txt"):

```
OPTION_ADVANCED_EXTERNAL_PROJECT_OPTIONS=%THEMIDA_FOLDER%\ExternalOp-
tions.txt
```

In your external options file, you specify the list of files to bundle.

```
OPTION_XBUNDLER_NUMBER_FILES=2

OPTION_XBUNDLER_FILE_NAME_AT_0=%INPUT_FILE_FOLDER%\MyFile.txt
OPTION_XBUNDLER_FILE_TYPE_AT_0=EXTRACT_NEVER
OPTION_XBUNDLER_FILE_VIRTUAL_PATH_AT_0=%APP_FOLDER%\MyFile.txt

OPTION_XBUNDLER_FILE_NAME_AT_1=%INPUT_FILE_FOLDER%\MyDll.dll
OPTION_XBUNDLER_FILE_TYPE_AT_1=EXTRACT_NEVER
OPTION_XBUNDLER_FILE_VIRTUAL_PATH_AT_1=%APP_FOLDER%\MyDll.dll
```

You can use any of the predefined constants to specify the path where the file to bundled is located, instead of using full paths, so you can easily move your project files and bundled files across different computers. You can refer to Special Constants in Input/Output file names 22.

1.8.4.2 I'm trying to embed a config file in XBundler (using "Never Extract to disk"option) and I want to modify that file in runtime, is that possible?

Please, notice that all embedded files in XBundler (with option "Never Extract to disk") are handled as read-only files, so you cannot modify them in runtime. If you want to modify a file that you are bundling, you have to select an extraction option for that file (instead of "Never Extract to disk"). For your specific scenario, where you want to write to an embedded file, you should set the option "Write if file not present" in the XBundler panel for that config file.

1.8.4.3 I'm using the option "Extract to disk" for several files that I'm bundling with XBundler. The files are extracted correctly under Windows XP but it fails under Vista and Windows 7. What's happening?

It seems that the problem is that the file cannot be created under Vista/Windows 7 due to UAC restrictions, which might be denying file creation if your application is not launched with an administrator manifest.

You have two options:

1) In the Extra Options panel 65, "Add a manifest from File" to run as administrator. This will force your application to be launched with admin's rights.

2) Go to the XBundler panel and right click on the XBundler files panel. Select the option "Add Root Folder --> USER_DOCS" (or any other special folder from the list). After that you can create a subfolder inside the added root folder. Example:

```
--> %USER_DOCS%
  |--> My Folder
     |--> File1.txt
     |--> File2.txt
```

The defined "Root Folders" are:

- APP_FOLDER: Folder where the protected application is located
- WIN_FOLDER: Windows folder (i.e: C:\Windows)
- WINSYS_FOLDER: Windows system folder (i.e: C:\Windows\System32)
- USER_DOCS: Current user documents folder
- LOCAL_APP_DATA: Local application data for the current user

- COMMON_APP_DATA: Common application data for all users
- TEMP_FOLDER: User temporal folder

1.8.4.4 I want to bundle my OCX in XBundler but not sure if it will work as my OCX needs to be registered in the system via regsvr32.exe

Latest versions of WinLicense supports ActiveX support in XBundler. You just need to go to the XBundler panel, add your desired OCX/DLLs, select the option "ActiveX Support" and protect your application

1.8.4.5 I want to XBundle my files but with relative paths to parent folders, so can I move my projects and files across computers and protect them from there?

In the XBundler panel, you can use specific constants to specify the location of the files to bundle. Please, refer to the "Original File Location" help section 48.

1.8.4.6 If I bundle some large graphics files and DLLs, is that going to influence the performance of my program?

Notice that everything that you embed with XBundler is encrypted all the time till it's required by your application. That is, if you load/unload your DLL or you open/close your graphic files, it will be decrypted and encrypted back. So, you could see some performance decrease if they are large files/DLLs and if they are opened and closed many times.

If you want to speed the above process, you can select the option "Maximize speed" in the XBundler panel 48.

1.8.4.7 I want to protect a DLL and bundle some data files using XBundler, but it does not work.

If you are having problems with embedded DLL, you can check the option "Exception Support in DLLs" in the XBundler panel 48.

1.8.4.8 Can I copy to disk a DLL that I have embedded with the option "Never Write to disk"

You can treat an embedded file exactly like if it were present on disk. If you want to extract a bundled file at any time, you can use for example the Windows API "CopyFile" and the embedded file will be copied to disk.

1.8.4.9 In my .NET application (test.exe) I want to to embed my .exe.config file (test.exe.config) with XBundler, but when I run my protected application (without the .config file) it does not see my embedded test.exe.config file.

Embedding .exe.config files in .NET applications is a bit tricky for some applications. You have to embed your original "test.exe.config" file with XBundler (option "Never extract to disk") but you have to ship your protected application with a "fake" or dummy "test.exe.config" file. This is because Windows checks the presence of a .config file before your application is launched. When your application reads information from the .config file, it will read it from the embedded one and not from the fake/dummy .config file. An example of a dummy .config file could be:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

</configuration>
```

1.8.4.10 Can I embed several EXE files inside XBundler and run them from memory, that is, without writing them to disk?

Sorry but execution from memory (without writing to disk) only works with DLLs and data files. Injection of application can only be done if you mark the option "extract to disk" for the .EXE file in the XBundler panel.

1.8.4.11 What about performance? I'm writing a filemanager which has a lot of access to local files.

All embedded files are encrypted inside the protected executable. When the file handle is closed, the file is encrypted again. Please, notice that opening and closing a file multiple times will make the whole process slower due to decryption/encryption.

In that case, we recommend that you protect checking the option "Maximize Speed" in the XBundler panel 48.

1.8.4.12 How are the files accessed from inside the protected application? May I call simply memo.lines.loadfromfile for example? If yes - does this mean that all accesses to files are filtered by XBundler?

Notice that accessing to bundled files should be TOTALLY transparent for you. XBundler must know how to handle all different ways to access to a file (even like "memo.lines.loadfromfile"). In case that you have compatibility problems with XBundler and your application, we are happy to work on it to fix your problem.

1.8.4.13 Can XBundler be used to bundle all of the DLL's and OCX's inside a protected DLL? Does there need to be an actual executable?

XBundler is mainly designed to operate from a protected EXE. So, you bundle all your DLLs to your EXE applications. Anyway, we added support to embed DLLs inside a protected DLL (like your scenario) but it could fail in some situations as reported from a customer (depending on the target application that use your DLL). For other customers, it works fine embedding DLLs inside a main DLL.

The best thing is that you try our DEMO version of WinLicense and see how XBundler goes with your scenario. For ActiveX protected DLLs, you might need to disable the option "Compress resources and encrypt resources" as Windows cannot register some DLLs when the resource section is compressed.

Anyway, if it does not work for you, it would be great if you can send us the test DLLs to reproduce the problem here, so we could add support for it.

1.8.4.14 Is it possible to use XBundler to bundle a console exe that I call run-time from my application?

Sorry but XBundler only works with DLLs and data files.

You can embed the console application inside XBundler and mark the option “extract to disk”, but you might not want that as the console application will be written to disk and viewable by your customers.

Data files and DLL can be embedded inside the protected application and can be used without writing them to disk.

1.8.4.15 Can I register my bundled DLLs with regsvr32.exe?

Sorry but bundled DLLs cannot be registered via "regsvr32.exe". Bundled DLLs are only visible for the protected application. As you can see, "regsvr32.exe" is an external process and won't be able to see the bundled DLLs.

Please, go to the XBundler panel 48, check the option "ActiveX support" and protect again. Your embedded DLLs will be correctly registered from the protected application. Notice that your application must be running with admin's rights the first time that is executed, so the registration of the bundled DLLs can be performed. You can add an admin manifest from the Extra Options panel 65.

1.8.4.16 I tried to bundle CHM file. Command which I use to open CHM file in my application below: ShellExecute(Application.Handle,'open','help.chm',nil,nil,SW_SHOWNOR MAL);

Sorry, but ShellExecute calls are not supported by XBundler. ShellExecute creates an external process and the embedded files in XBundler are only visible by the protected application, so the called process from ShellExecute cannot see the help file.

You need to call the Help file directly from your application (no using ShellExecute). Most programming languages have facilities to do that.

Please, notice that some programming languages open the Help file via FindFirstFile API, so, you need to enable the option "Hook FindFirst/Next File APIs" in the XBundler panel 48.

1.8.4.17 We are using a couple of DLLS and OCXS in our application, and I have tried unsuccessfully to use them with XBundler. What can I do?

If your DLL links with the Microsoft runtime libraries (MSVCRxx.dll), you could make a static linking with those libraries using the linking switch /MT (More information can be found at: http://msdn2.microsoft.com/en-us/library/abx4dbyh(VS.80).aspx) After that, compile again and it should work correctly with XBundler. Another solution is to check the option "Add Manifest from XBundler files" in the Extra Options 65 panel.

If the problem persists, it will be great if you send us your application and the DLLs that you plan to bundle which cause the problem, so we can reproduce the problem here. If you cannot send the unprotected application, it will be great also if you send us any test application that we can use to reproduce your problem.

1.8.4.18 I want to bundle my Visual Studio 2005 (or Visual Studio 2008) DLLs with XBundler but it fails to load them. Is that a known issue?

If your DLL is linking with the MSVC runtime libraries (MSVCR80 or MSVCR90), please go to the Extra Options 65 panel and check the option "Add Manifest from XBundler files".

1.8.4.19 I have inserted about 100 files to bundle. I want to select all of them and set for all of them "Extract always". Is that possible to do it without going one file at a time?

Yes, you can change the Extraction Mode for all selected files:

1) Select all the wanted files

2) Press:

- CTRL + 0 = "Never Write to disk"
- CTRL + 1 = "Extract always"
- CTRL + 2 = "Extract if not exists"
- CTRL + 3 = "Extract if older exists"
- CTRL + 4 = "Extract if different exists"

1.8.4.20 I have bundled several INI files with XBundler but when I try to access to them, they cannot be found. Other bundled files are working fine

If you are bundling INI files with the option "Never write to disk" you should check the option "Hook GetPrivateProfile APIs" and protect again.

1.8.5 Plugins

- When I protect my application using WinLicense DLL Control, it crashes if I compile my DLL with Delphi 2010, but it works fine if I compile my DLL with previous Delphi versions. What can I do? 280
- In my plugin, I'm using the function SecureEngineProcessHardwareId to change the hardware ID to something like "1234-56" but it does not work fine. 301
- I want to change the folder where the license needs to be present. I cannot use a fix location, but a different one for specific customers. Can I do it with DLL Control? 301

1.8.5.1 In my plugin, I'm using the function SecureEngineProcessHardwareId to change the hardware ID to something like "1234-56" but it does not work fine.

The problem is that you have to pass a hardware ID equivalent in format to the one in WinLicense. Something like:

1111-2222-3333-4444-5555-6666-7777-8888

Please, make sure that your own hardware IDs have the above format.

If you want a short hardware ID for your clients, then you can programmatically fill with zeros your short hardware ID to convert it to the format expected by WL. Example:

"1234-56" convert it (internally) to

"1234-5600-0000-0000-0000-0000-0000-0000"

Please, refer to the Remarks section in the SecureEngineProcessHardwareId 56 entry.

1.8.5.2 I want to change the folder where the license needs to be present. I cannot use a fix location, but a different one for specific customers. Can I do it with DLL Control?

Yes, you have to define the environment variable “WLDefaultLicenseFolder” with the path where the license file needs to be present. A good place to define the environment variable is at the SecureEngineInitialize 54 event.

Please, specify only the folder path and not the license name. Example:

```
SetEnvironmentVariable(“WLDefaultLicenseFolder”, “C:\MyApplication\MyLi-
censes”);
```

1.8.6 Customized Dialogs

1.8.6.1 I want to modify the string displayed by the Customized Dialog in runtime, displaying a more complex dialog. Is that possible?

Yes, you can do it using the Plugin system in WinLicense. You can handle the event SecureEngineShowCustomMessage 55 and modify the received message and display it as you desire.

1.8.6.2 I want to known how we can get and show a variable like %daysleft in my SecureEngineShowCustomMessage?

If you put %daysleft in the Customized Dialog panel for the specific message that you want, your event SecureEngineShowCustomMessage 55 will get that message with the "%daysleft" converted to the current days left. So, you just need to parse the message in case that you want to extract the current "days left".

1.8.6.3 Can we deactivate in any way the warning/error messages of WinLicense?

Please, go to the "Customized Dialog" panel. For messages MSGID0 to MSGID13 (Internal Protection Messages), you just need to edit the message and just leave a blank space in the message body. WinLicense will not display the messages which body consists in a single blank space.

For MSGID13 to MSGID35 (Trial Registration Messages) just select the specific message that you don't want to display and check the option "Displayed by SDK or not displayed" for that specific message.

1.8.6.4 I'm wondering whether it is possible to select the language for localization of WinLicense messages at runtime?

Yes, you can do it using the Plugin system in WinLicense. You can handle the event SecureEngineShowCustomMessage 55 and display the translation of the current received message.

1.8.6.5 If I want to show my dialogs (from inside my application) of days remaining and all other things, without using dialogs displayed after the WinLicense protection. Is that possible?

Please, go to the “Customized Dialogs” panel and you can see there all messages displayed by WinLicense. For each trial/registration message, you can tell WinLicense to display that message or not. That is, if for example you select the “MSG_ID_TRIAL_DAYS_EXPIRED” and you set it as "Disabled” that means that even if your trial has expired by days, WinLicense will NOT show any message and will KEEP your application running! So, you are the responsible of taking care of that event (by calling the WinLicense SDK API, like WLTrialDaysLeft in this example) and do the proper action when the trial is expired by days.

Remember that for each message, you can set if it will be handled by Winlicense or you (by selecting "Disabled" for the specific message). So, you just need to take care of those messages that you want to handle from your application and leave special message/events (like license corrupted, Stolen license key, etc) to be handled by WinLicense.

1.8.7 Trial

- Is there any option to reset the trial on a major update? I want to move my application from version 2.x to 3.x 305
- A user got the message: "The trial period of this application has been manipulated". What can I do to let him run my application again? 305
- I'm lost about how to use the Trial expiration. Please explain to me what stops someone from simply downloading a trial version, using it until it expires and then downloading another one? He can use it in trial mode forever? 306
- I can see that I can get the number of days left to know when my application will expire, but how can I know the exact hour/minute/second that it will expire? 306
- Can I store sensitive data in the Registry in an encrypted way? 306
- I see a lots of functions related with Trial/Registration, but I'm quite lost about which functions I have to use to check if my application is registered and if not, get the number of days left. 307
- I check "Run Time (execution)" in Trial Restrictions and I select to run the program for 2 minutes but the program is still running. Any ideas? 307

- Can I include different trial restrictions in a single application? 311
- If I use "Executions" as a method of protection, can I manually increase the counter by my program? 311
- I have created a .REG (registry) trial extension key, but it does not extend the trial. 311
- Can I lock (hardware id) to a single PC my application when it's running in Trial Mode? 312

1.8.7.1 Is there any option to reset the trial on a major update? I want to move my application from version 2.x to 3.x

In order to reset the trial on a new protected application, you have to change the "Trial Unique Key" (in the Software panel in the Winlicense GUI). The "Trial Unique Key" is an identifier to indicate where the trial will be stored in the system for the protected application.

If you re-protect your application and you don't change the "Trial Unique Key" the new protected version will continue the trial at the same point as your previous protected application. Or if you have two applications protected with the same "Trial Unique Key" they will both expire at the same time (when one expires the other one will expire too)

If you change the "Trial Unique Key" for your new protected application or for a secondary application that you protect, it will have a new location to store the trial information, so it has a fresh trial. That means that if you re-protect your application again with a different Trial Unique Key it has the same effect as a trial reset.

1.8.7.2 A user got the message: "The trial period of this application has been manipulated". What can I do to let him run my application again?

You can do any of the following options:

1) If you have enabled "Trial Extension" (In Trial Settings 32 panel) before protecting your application, you can send a trial extension key to your customer.

2) If you haven’t enabled the "Trial Extension" option, you can reprotect your application enabling that option and send him a trial extension key.

3) You can reprotect your application with a new "Trial Unique Key" (In Software panel). By doing that, your new protected application will have a fresh trial (as if the trial was reset on your customer’s PC).

4) You can send him an expiry license key, though this option is not suitable if your application works in different ways in trial and registered mode.

1.8.7.3 I'm lost about how to use the Trial expiration. Please explain to me what stops someone from simply downloading a trial version, using it until it expires and then downloading another one? He can use it in trial mode forever?

Suppose that you protect your application with 30 days trial (you set 30 days in the "Trial Settings" panel in WinLicense).

1) A person called "Bob" downloads and installs your application, so WinLicense stores in his system the current Trial status (30 days left).

2) After 30 days, Bob launches your application, so WinLicense detects that it's expired. When it's expired, WinLicense stores a special mark in the system which indicates that your application is expired.

3) Bob uninstalls, downloads and installs again your application.

4) WinLicense detects that the trial version is already expired, so, it won't allow more executions of Bob's PC.

If you want that a new version of your application clears the trial in Bob's PC (or any other PC) that already used a previous trial version, you have to select a new "Trial Unique Key" (in Software panel) for your application.This will store the trial expiration settings in a different place in Bob's PC, starting a clean trial period for your new version.

If you release a new version of your application and you don't change the "Trial Unique Key" in your new protected application, the trial will keep running as it was in your previous installed version (so, if Bob's PC had the previous version with 4 days left, the new version will have 4 days left).

1.8.7.4 I can see that I can get the number of days left to know when my application will expire, but how can I know the exact hour/minute/second that it will expire?

You can use the function WLRegExpirationTimestamp 152 to know the exact timestamp when your license will expire. If your application is running in trial mode, you can use the function WLTrialExpirationTimestamp

1.8.7.5 Can I store sensitive data in the Registry in an encrypted way?

Yes, to store simple 32-bit integers, you can use the Trial Counters API in WinLicense.

If you want to store long strings of data, you can use the "WLTrialStringWrite 132" and "WLTrialStringRead 129" functions. There are equivalents UNICODE functions also (WLTrialStringWriteW/WLTrialStringReadW)

1.8.7.6 I see a lots of functions related with Trial/Registration, but I'm quite lost about which functions I have to use to check if my application is registered and if not, get the number of days left.

Please, follow the next steps to check the status of your application and possible days/executions/date left.

1) You check if your application is in Registered or Trial mode (Using the function WLRegGetStatus 161)

2) If WLRegGetStatus returns that it's in Trial Mode, you check the Trial status (calling WLTrialGetStatus 126)

2.1) If WLTrialGetStatus returns trial valid, you can call the functions to know the trial days, executions (WLTrialDaysLeft, WLTrialExecutionsLeft, etc.)

3) If WLRegGetStatus returns that your application is registered, you can call the license expiration functions in case that you have expiry licenses (WLRegDaysLeft, WLRegExecutionsLeft, etc)

Summing up, you should never call a function to get the number of days/executions/date left without knowing the status of the application (via WLRegGetStatus and later WLTrialGetStatus)

1.8.7.7 I check "Run Time (execution)" in Trial Restrictions and I select to run the program for 2 minutes but the program is still running. Any ideas?

Please, can you go to the Customized Dialog panel and check if you have selected disabled for the message MSG_ID_TRIAL_RUNTIME_EXPIRED?

If you have set it to Disabled you are the responsible to close the application when the run time expires. You can check the run time expiration with the function "WLTrialGetStatus 126".

If you don't set the message MSG_ID_TRIAL_RUNTIME_EXPIRED to Disabled, Winlicense is the one that closes your application automatically.

1.8.7.8 I would like to detect whether my application is running in trial mode or registered mode programmatically. Which function do I have to use?

You just need to use the WLRegGetStatus 161 function. Please, refer to the WinLicense SDK for further information about this function.

1.8.7.9 I created a product v0.8 beta and set trial expiration to 15.06.2009. At 09.05.2009 I created new version of product (v0.9 beta) and set trial to 08.08.2009. Do I have to change the "Trial Unique Key" in the Software Panel

If you modify the Trial settings (in this case you are modifying the expiration date) and you already have a previous version protected (and released to public), you have to change the

"Trial Unique Key", so, for those users who are running an older version of your application, the new version will expire on the new date that you set.

1.8.7.10 When is the message "MsgID35 : Trial/Licenseing Management manipulated" displayed?

1) You have included days expiration or date expiration in your application and the user puts the clock back

2) The user has deleted several Registry keys where the Trial is stored. This usually occurs when the user runs one of those tools to reset the trial in shareware applications

3) You have protected your application with Trial Days (for example) and you reprotect again with Trial Executions without changing the "Trial Unique Key" (in Software Panel)

1.8.7.11 The extension key: how do they work? Do they work only if the trial license has expired?

Trial extension keys work with the trial expired and with not expired trial.

In case that it's not expired, it extends the current trial status. For example, if you create a trial extension key with 10 days trial then:

1) If your current trial is expired, you will get 10 days trial left.

2) If you currently have 5 days left, you will get 15 days left with the trial extension.

Notice that you can ONLY extend the trial settings that you selected in the "Trial Settings 32" panel in WinLicense before protecting your application.

1.8.7.12 I have protected my application with 30 days trial. I have put the clock back for 3 days and Winlicense reported the trial as expired (and that's what I want!). How can I make the trial working again for my application in my computer?

In order to reset the trial, you have different ways to do it:

1) Please, launch WinLicense, press "Software" in the tool bar menu, choose your application, and press the button "Reset Trial".

2) If you have protected the software in trial mode (when adding the option `OPTION_TRIAL_IS_DEBUG_MODE=YES` (in the Advanced Options 67 panel) you can just delete the registry key `HKEY_CURRENT_USER/SOFTWARE/WinLicense/WLdebugTrial`

3) If you have protected with the option "Allow trial period to be extended (with a valid key)" checked (in Trial Settings 32 panel), you can create an extension key that will extend the trial period for your protected application

4) The last way is to create a new "Trial Unique Key" (in the Software 363 Panel) and protect again your application. That will store the Trial period in another place in the system, with the same effect as if the trial was reset (in will reset the trial in any of your clients).

1.8.7.13 Is it possible to have such trial limitations in a single application: 10 executions and 30 days? What we have in such configuration - application became expired after its first execution.

Sure, you can combine both of them (and many other trial restrictions)

Notice that if for example you have protected your application with "Trial Executions" and later you protect your application with "Trial Executions" + "Trial days", your trial will become expired/corrupted. That is because your current "Trial Unique Key" (in Software panel) in your application has been installed in the target machine to run only with Trial Executions, so, if you later re-protect with Trial Executions + Trial Days, you get trial corruption.

In order to solve that, you have to change your "Trial Unique Key" and reprotect again your application. So, you will have your protected application being able to expire in executions and/or days.

1.8.7.14 How to calculate "days left" for license that must expire on specified DATE? I tried to use function WLRegDaysLeft(), but it returns -1 for license with specified expiration date.

The function WLRegDaysLeft 144 returns the days left when you create a license with trial **days** expiration (not date expiration option). If you have created a license with date expiration and you want to explicitly get the number of days left until that date, you can use the function WLRegDateDaysLeft 142.

1.8.7.15 Can I protect putting trial days expiration with the DEMO version of WinLicense? How difficult is to reset the trial?

You can use all trial options in the DEMO version of WinLicense.

Notice that the Trial period in WinLicense DEMO is protected in “debug mode”, that is, the trial can be easily reset just deleting the key: **HKEY_CURRENT_USER/Software/WinLicense/WLdebugTrial**

This feature is to allow developers to test their applications protected without putting unnecessary information in their system about the trial.

In our registered version of WinLicense, the protected application will store the trial information in hidden places in the system to avoid being reset by attackers.

1.8.7.16 I have protected my application with Trial Settings. When I uninstall and install my application again, the Trial continues at the same point that it was before uninstalling it. How can I reset the trial info from my computer?

If you remove your program and you install it again, the trial is still stored in the system. This is a feature in WinLicense to avoid that your users reset the trial in your application by uninstalling and installing again your application

If you want to reset the trial of your new protected instance, you can just change the "Trial Hash" in the Software panel 363.

1.8.7.17 What are these Custom Counters? How can I use them?

Custom counters are available for different developer needs. They can be freely used to keep the count of an expirable resource in trial versions. For example, an image editing application needs to be restricted to save only 100 edited images in trial version. A custom counter could be used in this case: after each save operation, the corresponding counter must be increased.

WinLicense offers 3 basic functions to manipulate custom counters.

To get extended information about the Custom Counters WinLicense API, please refer to the following functions:

· WLTrialCustomCounter 107

· WLTrialCustomCounterDec 108

· WLTrialCustomCounterInc 110

1.8.7.18 I'm going to release a new version of my application with a bug fix, but I want the trial period to continue as it was for each client. How can I keep the trial period for my new protected application?

To do this the same Trial Hash must be used for both old and newer versions. If you are creating a new Software in WinLicense for your new version, just copy the Trial Hash from old version to the new one using the Software Manager 363 panel.

1.8.7.19 How can I clear the trial information when I release a new version?

The solution is to use another Trial Hash for the new version. This can be done from the Software Manager 363 panel. After selecting a new trial hash, the new version needs to be protected again.

1.8.7.20 My trial extensions keys do not seem to be working to extend the trial period in my application

Please, make sure that you are following these requirements:

· You have enabled the option "Allow trial period to be extended" (in the Trial Settings 32 panel)

· Either "Extension file name" or "Extension Registry location" is enabled in the Trial Settings panel.

· You are extending a restriction that was included when your program was protected. That is, if you have restricted your application by number of executions, ONLY the number of executions can be extended with the trial extension keys.

1.8.7.21 Can I include different trial restrictions in a single application?

Yes, WinLicense offers a flexible trial system that is able to include any combination of the trial restrictions of your application. Please, refer to the Trial Settings 32 panel for this.

1.8.7.22 If I use "Executions" as a method of protection, can I manually increase the counter by my program?

Yes, you can use the function WLTrialExtendExpiration 120, where you can extend the expiration of your application. Also, you can use create a trial extension key in case that you have enabled the "Trial Extension" option (in the Trial Settings 32 panel) when your application was protected.

Please, notice that you can only extend the trial expirations that you have selected in your application. That is, if you have protected your application with expiration by executions, you can only extend the executions number from WLTrialExtendExpiration or using trial extension keys.

1.8.7.23 I have created a .REG (registry) trial extension key, but it does not extend the trial.

Please, make sure that:

1) You have enabled the option "Allow trial period to be extended" and "Extension Registry location" (in the Trial Settings 32 panel)

2) You are not creating a trial extension key with "level" bigger than the maximum allowed in the field "Maximum extensions allowed"

3) If you have already created a trial extension with "Level 1" for example, and you want to extend it again, you have to create a trial extension with "Level 2"

4) If your application is protected with expiration by days (for example), then you can ONLY extend expiration by days in your trial extension keys.

5) When you double click on the .reg extension key, please, check the information is stored in the Windows Registry

1.8.7.24 Can I lock (hardware id) to a single PC my application when it's running in Trial Mode?

Sorry but applications in Trial Mode cannot be locked to a specific computer. If you specify a 15 days trial period for your application, it will run during 15 days on ANY computer.

If you want to limit time your application and also lock it to specific machines (and not locked to other machines), you have to create licenses for your application.

You can create a license (with 15 days restriction, for example) and set the "HardwareID" field in the license to restrict the license to a specific machine, so, it can only be used on a machine.

If you create a license, leaving the "HardwareID" field as blank, the license will run on ANY computer.

Summing up, if you want to limit time your application and restrict the execution in some machines, you have to create licenses with expiration (and specifying the HardwareID for those licenses that you want to lock to a specific machine)

1.8.8 Registration

1.8.8.1 Smartkeys

1.8.8.1.1 When I try to generate a Static SmartKey with more than 255 days expiration, the generate key does not have more than 255 days expiration.

Please, notice that Static SmartKeys days expiration are limited to 255 days to keep the SmartKey string as short as possible. We recommend using Dynamic SmartKeys which do not have that limitation and also offer more security than Static SmartKeys.

1.8.8.1.2 I have enabled Dynamic Smartkeys and Registry keys in the "Registration" panel. The problem is that if I create a .reg key directly and I later call WLRegGetDynSmartKey I don't get the Smartkey. What's wrong?

WLRegGetDynSmartKey 154 only works for Dynamic Smartkeys that you create explicitly. That is, if you install a file or registry key directly, you cannot get the original Dynamic Smartkey (because it's not in fact a Dynamic Smartkey). Please, keep in mind that you also have the

"Custom Data" field in the license, where you might want to store similar information that you want to retrieve in runtime (via WLRegGetLicenseInfo)

1.8.8.1.3 I am calling WLRegSmartKeyCheckW (UNICODE) and generating licenses under the "License Manager" in WinLicense, but all SmartKeys are rejected. Why?

Please, make sure that when you are going to generate the license (entering the User's details) from the License Manager, you check the option "UNICODE License" in the "Miscellaneous" section.

1.8.8.1.4 I have included all user information (Name, Company, Custom Data) inside a Dynamic SmartKey. How do I call WLRegSmartKeyCheck if I just have a single SmartKey string? I mean, the function requires 4 parameters and I only have one (the SmartKey string)

If you embed all user information (Name, Company, Custom Data) inside the Dynamic Smart Key, you just need to pass the SmartKey string to WLRegSmartKeyCheck and WLRegSmartKeyInstallToFile/WLRegSmartKeyInstallToRegistry. Example:

```
if (WLRegSmartKeyCheck(NULL, NULL, NULL, pMySmartKey))
{
    WLRegSmartKeyInstallToFile(0, 0, 0, pMySmartKey);
}
```

1.8.8.1.5 How can I obtain the license key as a string once the application has been activated using a SmartActivate key?

For Static SmartKeys, there is no way to obtain the SmartKey string that registered the application.

In latest Winlicense versions, we have introduced the new "Dynamic SmartKeys" which offers more security and flexibility than "Static SmartKeys". You can also obtain the SmartKey string once the application is registered using the function WLRegGetDynSmartKey 154.

Summing up, in order to get the SmartKey string, you have to use Dynamic SmartKeys and call the function WLRegGetDynSmartKey.

1.8.8.1.6 When I call WLRegLicenseCreationDate from inside my protected application, I cannot get the creation date for my SmartKeys.

Notice that old SmartKeys (Static SmartKeys) don't have support to embed the creation date inside of the SmartKey. You have to use the new "Dynamic SmartKeys" or the common "File keys, Registry Keys or Text Keys".

Notice that if want to embed the creation date inside your generated license, you have to specify it explicitly when you are going to generate a new license. From the "License Manager", you just need to check the option "Store Creation Date" before generating a new license.

If you are creating licenses from your own license generator, you have to use the new functions (WLGenLicenseFileKeyEx 200, WLGenLicenseRegistryKeyEx 207, WLGenLicenseTextKeyEx 215, WLGenLicenseDynSmartKey 218) which accepts a parameter "LicenseFeatures" that specifies the features of the license to generate.

The definition of that structure is the following:

```
typedef struct _sLicenseFeatures
{
   unsigned cb;
   unsigned NumDays;
   unsigned NumExec;
   SYSTEMTIME ExpDate;
   unsigned CountryId;
   unsigned Runtime;
   unsigned GlobalMinutes;
   SYSTEMTIME InstallDate;
   unsigned NetInstances;
   unsigned EmbedLicenseInfoInKey;
   unsigned EmbedCreationDate;
} sLicenseFeatures;
```

You can see the last field "EmbedCreationDate". You have to set that field to "1" if you want that your generated license will include the creation date.

1.8.8.1.7 I have created a SmartKey for my customer, but when he enters the SmartKey and I call WLRegSmartKeyCheck, the key is always invalid. For other customers, my generated keys work fine. Why?

If some keys are working but not others, it might be because you have locked your license to a machine (hardware ID) and the user is not inserting the license in the specific PC (with different hardware ID)

1.8.8.1.8 When I generate a SmartKey for a single user several times, I can see that the generate SmartKey is different. Is that correct?

Yes, the SmartKey is totally different each time that you generate it. Internally, it has an encryption key which makes the SmartKey look so different each time (even if the customer information is exactly the same)

1.8.8.1.9 I'm still lost about how to use SmartActivate keys, can you give me some examples?

As you might know, an application can be registered via a file key or a Registry key. SmartActivate keys is another way to register an application but it's basically the same as a file or Registry key, though we have to make use of some APIs in order to install the SmartActivate key as a file or Registry key.

The steps are the following:

1. In the Registration panel, we have to check the option "Enable SmartActivate System for user-side generated keys"

2. You have to choose if you will finally install the SmartActivate key as file or Registry key, so we check the option "Single File" (In the Registration panel) if you plan to finally install the SmartActivate key as a file key

3. In your application, you have to implement a dialog where you allow a user to register the application via SmartActivate keys (See examples in the WinLicense examples folder)

4. Protect your application

5. Let's create a SmartActivate key from the WinLicense License Manager. You have to give your client the SmartActivate key information in order to register your application

6. When your customer inserts the SmartActivate key in the dialog where you allow the insertion of SmartActivate keys, you have to call the API WLRegSmartKeyCheck 178 and if it returns TRUE, you have to install the SmartActivate key as file (or Registry) using WLRegSmartKeyInstallToFile 182 (or WLRegSmartKeyInstallToRegistry 189)

7. Restart your application (you can call WLRestartApplication 258) and it should run as registered!

1.8.8.1.10 What are SmartActivate keys?

SmartActivate Keys, also known as SmartKeys, are a special type of license keys accepted by WinLicense. The SmartKey appears in the following manner:

Name: John Smith
Company: J.S. Software Development
Custom Data: MODE: Advanced
Key: 286D0394-87716B52-BC59FA9A-CAA8DE87-4881A155-BF450169-C590

It is very popular now to use such keys when almost all registration data comes as plaintext and the key has a checksum for data and license restrictions. The SmartKey is quite short and contains only printable characters so it will be easy to copy/paste for sending through e-mail, instant messengers etc.

1.8.8.1.11 In last 2 weeks we got a lot of customers who was not able to register their products under Vista. We use SmartKeys with hardware id. Please advise.

You have not specified if you are installing the SmartKey as a Registry key or as a File (using either WLRegInstallSmartKeyToRegistry or WLRegInstallSmartKeyToFile). Those settings are in the Registration panel.

For Registry, you should specify the HKEY_CURRENT_USER to avoid problems with restricted accounts or admin users running with UAC enabled.

For File, you should use any of the new constants to put the file license into a writable folder for restricted users (or admin in UAC mode). You can specify a defined folder name as described here 332.

1.8.8.1.12 I want to mark as stolen (or ban) a specific SmartKey for a customer, but I don't want to invalidate all licenses for that customer. How can I do it?

Yes, you can do it with WinLicense. Please, refer to the following section 97.

1.8.8.1.13 I'm using custom data to store extra license information but I don't want the user to insert custom data or show him these values. I want the custom data to be part of the SmartKey number, so, he cannot see it.

For Static SmartKeys the user needs to insert the Customer Name + Company + Custom Data + SmartKey serial (you can avoid passing the "Name" or "Company" or "Custom Data", just passing a NULL string). The HardwareID does not need to be entered by your customer as it's taking internally from the current machine. As you can see, the Custom Data must be explicitly entered by your customer or not included it, but it cannot go as part of the SmartKey serial.

If you use the new Dynamic SmartKeys, you can embed all user information (Name, Company, Custom Data) within the generated SmartKey. That is, the generated SmartKey will be much longer than the old Static SmartKeys but your customer just receives a single SmartKey (which contains all his information inside)

1.8.8.2 Network Instances

- I have created a license with 10 network instances. When the application is launched 2 times in the same computer, I get 2 instances running. Is it possible to count only by computers? 318
- I have created a license with Network Instances for one of my customers. He has started complain that he gets some noticeable traffic on the network produced by my protected application. Is it possible to decrease that network traffic? 318
- I'm just testing your application with the network options switched on. I have a server application which is working fine but, do the clients need a licence file? 318
- Can I limit a license to a number of instances inside a network? 319

- Can I include hardware locking to a license which is also limited to a number of instances inside a network? 319
- I want to generate a SmartKey with network instances, but the function WLGenLicenseSmartKey does not include the network instances parameter. How can I do it? 319
- Can I use the Network Instances feature on my protected DLL? 320
- How do I display the IP addresses that have been retrieved with WLRegNetInstancesGetClientsIp? 320

1.8.8.2.1 I have created a license with 10 network instances. When the application is launched 2 times in the same computer, I get 2 instances running. Is it possible to count only by computers?

Yes, you can just count by computers, so, if you launch several instances on a single computer, it will just count as one instance on the network.

Please, go to the Advanced Options panel and add the following line:

```
OPTION_ADVANCED_NETWORK_INSTANCES_COUNT_BY_COMPUTERS=YES
```

1.8.8.2.2 I have created a license with Network Instances for one of my customers. He has started complain that he gets some noticeable traffic on the network produced by my protected application. Is it possible to decrease that network traffic?

If you create a license with network instances, WinLicense will check the network periodically to know how many instances are running in the local network. If you want to decrease the network traffic, please, decrease the number of milliseconds for the "wl_net_timeout" and "wl_net_wait_ms" options. You can find more information here 101.

1.8.8.2.3 I'm just testing your application with the network options switched on. I have a server application which is working fine but, do the clients need a licence file?

To use network licenses in Server-Client mode, please, make sure you follow the next steps:

1) Protect your application and create a hardware locked license (locked to the Server Machine) and set in the license the number of network instances allowed

2) Place the protected application in the Server machine with the generated license.

3) Run the application in the Server machine (don't close it)

4) Now, other computers on the network can also run the protected application while there are network instances available. You can copy the protected application and the generated license on each specific client machine or create a shared folder in the server, where clients can see the protected application and launch it from there.

5) If instances in the network cannot be launched, please, refer to the possible event errors here 101.

1.8.8.2.4 Can I limit a license to a number of instances inside a network?

Yes, you just need to go to the License Manager in WinLicense and when you are going to generate a license, you have to set the option "Network Instances" with the number of instances that you want to limit.

You can also create your licenses with "Network Instances" restrictions from your license generator (using the WinLicense SDK to generate licenses 194).

1.8.8.2.5 Can I include hardware locking to a license which is also limited to a number of instances inside a network?

Yes, in fact, the new Network Instances module in version 3.0 forces you to hardware lock the network instances license. You have to generate the license locked to the machine that will act as the "server". That server machine needs to run first the application and after that, other client machines can start the application.

1.8.8.2.6 I want to generate a SmartKey with network instances, but the function WLGenLicenseSmartKey does not include the network instances parameter. How can I do it?

The current format of the SmartKeys does not support Network Instances specifically (it is supported in the newer Dynamic SmartKeys). But there is a workaround to limit a SmartKey to a specific number of instances inside a network.

If you are going to generate a SmartKey from inside the "WinLicense License Manager" and you put a network instances limit in a SmartKey, WinLicense will explicitly put a special string in the beginning of the Custom Data field to identify the network instances limit in the SmartKey.

If you want to create SmartKeys with network instances limit from outside the WinLicense License Manager (via WLGenLicenseSmartKey 221), you have to enter the network instances limit as follow: *NL*%4x, that is, "*NL*" + four bytes number in hexadecimal format, which specifies the number of instances in a network. If for example, you want to generate a license with the "MODE:Test" string in the custom data and you also want to limit the SmartKey with 12 instances in a network, you have to pass the following string in the Custom Data parameter to the WLGenLicenseSmartKey API as follow:

*NL*000CMODE:Test

Where "*NL*000C" means 12 instances (000C hexadecimal = 12 decimal)

Note: When you call the API WLRegGetLicenseInfo 156, WinLicense will return the Custom Data information from the license but without the "*NL*%4x" information.

1.8.8.2.7 Can I use the Network Instances feature on my protected DLL?

There are some restrictions about that. When applying network instances on a protected DLL, the protection cannot determine in boot time (before your DllMain is executed) the number of instances running. This is because the server/client instance is not fully registered until your DllMain is executed.

As the protection boot loader cannot determine the number of running clients until your DllMain is executed, you should call the function WLRegNetInstancesGet 167 to know the number of running instances and compare it with the maximum number of network instances allowed (WLRegNetInstancesMax 168). You have to call those functions after your DllMain has been executed.

1.8.8.2.8 How do I display the IP addresses that have been retrieved with WLRegNetInstancesGetClientsIp?

The following code example might help you to know how to display the IP addresses from the function WLRegNetInstancesGetClientsIp 168.

```
    const int MAX_IPS = 100;
    WL_IP_ADDRESS buffer_ips[MAX_IPS];

    int number_ips = WLRegNetInstancesGetClientsIp(buffer_ips, MAX_IPS);
    TCHAR strOutput[MAX_IPS * 40] = { 0 };

    for (int i = 0; i < number_ips; i++)
    {
        TCHAR ip[256];

        if (buffer_ips[i].is_v6)
        {
            _stprintf_s(ip, 256, _T("%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:%02x%02x:
%02x%02x:%02x%02x"),
                buffer_ips[i].i16,
                buffer_ips[i].i15,
                buffer_ips[i].i14,
                buffer_ips[i].i13,
                buffer_ips[i].i12,
                buffer_ips[i].i11,
                buffer_ips[i].i10,
                buffer_ips[i].i9,
                buffer_ips[i].i8,
                buffer_ips[i].i7,
                buffer_ips[i].i6,
                buffer_ips[i].i5,
                buffer_ips[i].i4,
                buffer_ips[i].i3,
                buffer_ips[i].i2,
                buffer_ips[i].i1);

            lstrcat(strOutput, ip);
        }
        else
        {
            _stprintf_s(ip, 256, _T("%d.%d.%d.%d"),
                buffer_ips[i].i4,
                buffer_ips[i].i3,
                buffer_ips[i].i2,
                buffer_ips[i].i1);

            lstrcat(strOutput, ip);
        }
        lstrcat(strOutput, _T("\r\n"));
    }
    MessageBox(hDlg, strOutput, _T("Client IPs"), 0);
```

1.8.8.3 Common

- I want to specific the output file name of the generated SmartKey in the "License Manager". Can I do that? 326
- I want that my protected software asks for a license when it runs. If there is no license I don't want that my application starts. Can I do it with WinLicense? 326
- I want to add an 'add-in' or 'plug-in' to my software that requires payment. Is there anything built-in to WinLicense to support that? 327
- I'm creating licenses using my own external key generator as an EXE file. I have seen that my key generator (.EXE) depends on WinLicenseSDK.dll to work. Is there any way to avoid the linking with WinLicenseSDK.dll? 327
- I have created a license with expiration date. What happens if my user leaves the application running on memory for months, even after the expiration date occurs? 327
- How to simulate the CustomData value when application running in non-protected state? I would like to use WinlicenseSDK.ini to specifiy the property. 328
- My licenses are locked to a USB pen-drive. How can I know if the user unplugs the pen-drive after my application has started? 328
- What's the difference between the macro REGISTERED_START and REGISTEREDVM_START? Which one should I use? 266

1.8.8.3.1 I want to specific the output file name of the generated SmartKey in the "License Manager". Can I do that?

If you edit or create a new Software, in the lower section "Custom Values", you can enter different options that are mostly related with key generation.

You can edit the key "LicenseFileTextName" and put there the default name for your SmartKeys.

1.8.8.3.2 I want that my protected software asks for a license when it runs. If there is no license I don't want that my application starts. Can I do it with WinLicense?

Yes, just follow the next simple steps:

1) In the "Registration panel" you have to check the option "This application can be registered..."

2) Select the key type that you expect for your application (File License, Registry License, SmartKey...)

3) Check the option "Application only runs when Registered (Requires a key to run)"

Now, if you launch your application and a license key is not present, your protected application will display the message MSG_ID_LICENSE_REQUIRED_RUN (in the Customized Dialog panel). You can edit that message to display what you want. If you want to display your own forms or nice dialogs, you can use the Plugin system 52.

1.8.8.3.3 I want to add an 'add-in' or 'plug-in' to my software that requires payment. Is there anything built-in to WinLicense to support that?

If you want to use a separate license for each plugin, then you just need to put a different "License Hash" (in the Sofware panel 363) for each plugin that you protect. You can select the expected license name for each plugin in the Registration panel 36.

You can also use the same license to register all your plugins and use the "Custom Data" field to specify which plugin is supported. You can enter any text (up to 8000 chars) in the "Custom Data" field and read it in runtime via "WLRegGetLicenseInfo 156".

For example, the custom data field could contain a list of supported plugins names, in runtime you call WLRegGetLicenseInfo and you parse the "Custom Data" array to load the required plugins.

1.8.8.3.4 I'm creating licenses using my own external key generator as an EXE file. I have seen that my key generator (.EXE) depends on WinLicenseSDK.dll to work. Is there any way to avoid the linking with WinLicenseSDK.dll?

The key generation code is implemented inside WinLicenseSDK.dll, that's why you need to have WinlicenseSDK.dll along with your key generator to work.

The key generator functions are different from the Trial/Registration APIs that WinLicense offers. The Trial/Registration APIs do not require WinLicenseSDK.dll once that the application is protected, because WinLicense redirects those APIs to its internal protection code to make the real work, so you can deliver your protected application to your customers without shipping WinLicenseSDK.dll.

If you don't want to link with WinLicenseSDK.dll, probably because you are using an external company service to generate licenses for your protected application, we have a C ANSI source code generator that you can easily compile to generate licenses for your protected applications. Of course, the C ANSI code does not require the WinLicenseSDK.dll to work and you can compile it to work under any platform. This C ANSI code is available for any of our customers who request it.

1.8.8.3.5 I have created a license with expiration date. What happens if my user leaves the application running on memory for months, even after the expiration date occurs?

1) Please, go to the "Customized Dialog" panel and check if the MSG_ID_LICENSE_DATE_EXPIRED is set as Enabled or Disabled.

2) If you have set it as Enabled, WinLicense will stop the execution on your application in memory when that date arrives and it will display the defined message for that dialog.

3) If you have set it as Disabled, WinLicense will continue running your application normally even after your license expires. You have to call periodically the SDK (WLRegGetStatus 161) to know if your license is already expired or not, so you can do your proper actions in case that it expires.

1.8.8.3.6 How to simulate the CustomData value when application running in non-protected state? I would like to use WinlicenseSDK.ini to specifiy the property.

Please, open the "WinlicenseSDK.ini" file with Notepad.exe (or any text editor) and you can find the following line:

WLRegGetLicenseInfo = Peter Brian/Company Name/pbrian@companyname.com

The "/" separator in WLRegGetLicenseInfo corresponds to Name/Company/CustomData.

In the above example, the custom data is "pbrian@companyname.com"

Notice that you can change the "Separator" char by specifying it inside the WinLicenseSDK.ini file. Just open WinlicenseSDK.ini and edit the "Separator" entry. Example:

```
[WinLicenseSDK]

; Separator for fields in "WLRegGetLicenseInfo"

Separator = /
```

....

1.8.8.3.7 My licenses are locked to a USB pen-drive. How can I know if the user unplugs the pen-drive after my application has started?

You have to check the option "Detect Unplug" (in the Hardware Lock panel 40). When the USB is disconnected (after a few seconds) WinLicense will display the MSG_ID_USB_LOCKING_UNPLUGGED message (from the Customized Dialog panel 43)

1.8.8.3.8 For specific customers, I ship a license with days expiration. For other customers, I ship a license with date expiration. How can I know if the current license has expiration by days or date?

You can use the function WLRegGetLicenseRestrictions 158 to know the expiration restrictions in the current license.

1.8.8.3.9 I can see that I can get the number of days left to know when my application will expire, but how can I know the exact hour/minute/second that it will expire?

You can use the function WLRegExpirationTimestamp 152 to know the exact timestamp when your license will expire. If your application is running in trial mode, you can use the function

WLTrialExpirationTimestamp 118

1.8.8.3.10 I want to control downloads (updates) of my software. My clients will have a license with one year of updates. After one year, updates are not allowed but he can continue using the software. Can WinLicense do this?

With WinLicense, you can create your license (embedding User Name, Company and Custom Data) and license restrictions (like days limit, execution limit, date limit, etc). You can read the license information using the API WLRegGetLicenseInfo 156

Basically these are the steps:

1) You create a license for your customer with 1 year expiration

2) When the license expires (after 1 year) your application goes into Trial state (so, in this state you don't allow updates for example). You can check if your application is running as Trial or Registered via the function WLRegGetStatus 161

3) You sent a new license (re-create license) for your customer after he makes the payment and it will be extended for another year (Each license has an unique ID, so if you recreate a new license, even with the same user information, the expiration will be set according to the new license)

4) Don't forget that if you want to control the status of your application (registered, expired, trial, etc) you have to use the WinLicense SDK APIs and you have to Disable the specific expiration events/messages from the Customized Dialog 43 panel. Please, check also the following KB entry 303

1.8.8.3.11 Is a generated file key compatible for future changes of my application and future updates of WinLicense?

As long as you don't change the "License Hash" for your application (in the Software 363 Panel), all your generated licenses will work fine with future versions of your application (even when protecting with upcoming Winlicense versions).

If you change the "License Hash", you will have to generate new licenses for your protected software.

1.8.8.3.12 How can I specify as license file name the name of my current Software? So, I can reuse the same project for several Software.

Please, go to the Registration 36 panel and use the constant “%SOFT_NAME%” as part of your license key name in the “File License” edit box. Example:

```
%SOFT_NAME%.dat
```

1.8.8.3.13 I see a lots of functions related with Trial/Registration, but I'm quite lost about which functions I have to use to check if my application is registered and if not, get the number of days left.

Please, follow the next steps to check the status of your application and possible days/executions/date left.

1) You check if your application is in Registered or Trial mode (Using the function WLRegGetStatus 161)

2) If WLRegGetStatus returns that it's in Trial Mode, you check the Trial status (calling WLTrialGetStatus 126)

2.1) If WLTrialGetStatus returns trial valid, you can call the functions to know the trial days, executions (WLTrialDaysLeft 114, WLTrialExecutionsLeft 116, etc.)

3) If WLRegGetStatus returns that your application is registered, you can call the license expiration functions in case that you have expiry licenses (WLRegDaysLeft 144, WLRegExecutionsLeft 149, etc)

Summing up, you should never call a function to get the number of days/executions/date left without knowing the status of the application (via WLRegGetStatus and later WLTrialGetStatus)

1.8.8.3.14 When I call the function WLRegGetLicenseInfo, it works fine when the license is not expired. BUT when the license is expired, the user name/company are filled with garbage data. What's happening?

Notice that this is not a bug, but the current design of the licensing system in WinLicense. When a license is expired, the application goes into Trial Mode. As your license is expired, your application is in Trial Mode, so the function WLRegGetLicenseInfo 156 will return FALSE as there is no valid and not expired license installed. So, you will not get the license information when the license is expired.

You should first call the function WLRegGetStatus 161 to know if your application is registered with a valid and not expired license. When you know that your application is correctly registered then you can call the function WLRegGetLicenseInfo.

To display the registration information once that your license is expired, you could store the license information in any place in the system when the application is registered (and not expired), so you can read it at a later time. If you want to store information in an encrypted way, you can use our functions "WLTrialStringRead 129/WLTrialStringWrite 132" which are available in either Trial or Registered state.

1.8.8.3.15 I want to mark a license as stolen using WLRegDisableCurrentKey but not sure how to proceed. My application will communicate with my web server to check if the license is stolen. Please, help!

Notice that the function WLRegDisableCurrentKey 147, disables the license installed in the current computer.

If your application connects to your web server, you could implement something like this:

1) Your application starts and it sends to your server the registration information (WLRegGetLicenseInfo 156)

2) Your server replies if license status is OK (not stolen)

3) If license is reported as stolen from your server, then the server returns a special code to your application (like ExitCode = stolen_lic)

4) If your application receives "ExitCode = stolen_lic", then application executes "WLRegDisableCurrentKey" and the license will be banned in the current computer

1.8.8.3.16 How do I register my application when it's expired? I see that WinLicense displays the expiration messages and exits.

The key to keep your application running even if it's expired is in the Customized Dialog panel. In the Customized Dialog panel you can tell Winlicense which messages you are handling by yourself and which ones are going to be handled by Winlicense.

Suppose that your application is protected with 30 days trial expiration. If you go to the Customized Dialog 43 panel, select the MSG_ID_TRIAL_DAYS_EXPIRED and set it as Diabled (by right-clicking on it). Now you are telling Winlicense not to display that message when the application expires and keep your application running. So, you should check if your application is expired or not using the Winlicense SDK (in this case, you should call the function WLTrialGetStatus 126). If your application is expired, you can do what you consider, like displaying a registration form, limit functionality in your application, etc.

Notice that for each message that you plan to handle in the Customized Dialog panel, you should make sure that are really handling that message by calling the WinLicense SDK. A common mistake is to select all messages and for all of them you set them as "Disabled" and you are not checking with the WinLicense SDK when those events occurs (like a corrupted license, if the license has a wrong hardware ID, etc.) and your application will continue running normally when those "events" occurs.

1.8.8.3.17 Can I integrate the license generation for paying customers into our web site by calling it as a web service or CGI or just as a utility from the web application?

If your server runs under Windows platform, you can use the WinLicenseSDK.dll and call the license generator functions (please, refer to the help file for extended information about each License Generator function available)

If your server runs under Linux, UNIX, etc, then you can use our C ANSI code generator, which you can compile to generate licenses under any platform. Please, contact us for more information at info@oreans.com

1.8.8.3.18 In the Registration panel, I would like to specify a path (like user documents) in the "Single File" field, so my licenses are expected to be located there. Is that possible?

Yes, in the "File License" field (in the Registration 36 panel), you can specify one of the defined WinLicense directory constants to place your licenses in different Windows common folders, like:

- **%USER_DOCS%** (or %userdocs%) : Specifies the current user documents folder (!My Documents).

- **%USER_APP_DATA%** (or %userappdata%) : Specifies the current user application data folder (!{user name}\Application Data)
- **%COMMON_APP_DATA%** (or %commonappdata%) : Specifies the common application data for all users (!All Users\Application Data)

- **%LOCAL_APP_DATA%** (or %localappdata%) : Specifies the local application data for the current user (!{user name}\Local Settings\Application Data (non roaming))

- **%WIN_FOLDER%** (or %windir%): Specifies the Windows folder

- **%TEMP_FOLDER%** (or %tempfolder%): Specifies the user temporal folder

- **%PUBLIC_DOCS%** (or %publicdocs%): Specifies the public documents folder

Notice that the above constants are case sensitive. You can specify sub-directories with any of the above constants. Example:

%USER_DOCS%\MyApplication\Licences\license.dat

If you are installing a file license from a SmartKey (via WLRegSmartKeyInstallToFile 182) or from a Text key (WLRegNormalKeyInstallToFile 172) the generated file will be placed in the expected path that you put in the "File License" field (in the Registration 36 Panel)

1.8.8.3.19 I'm using the License Generator functions in my application. When I protect my license generator, it still requires the WinLicenseSDK.dll file!

When you protect your application, WinLicense removes the linking with the WinLicenseSDK.dll, so, WinLicense will return the real values for the SDK functions.

This is the case for all WinLicense SDK functions except the License generator function. The code to generate a license, it's implemented inside the WinLicenseSDK.dll itself, so, for the License Generator functions, the linking with the WinLicenseSDK.dll is not removed.

Summing up, your license generator can be protected, but it will require the WinLicenseSDK.dll to be able to generate licenses for your application. Anyway, notice that it's not common (or necessary) to protect your License Generator as normally you are the only one that has access to it.

1.8.8.3.20 How would I go about implementing a system where keys only work for updates 12 months after the software is purchased? After that the customer has to renew the license otherwise the updates won't work with the existing keys?

You could create a license with 12 months expiration (365 days expiration) and you control the event when the license expires in the Customized Dialog 43 panel and for the message MSG_ID_LICENSE_DAYS_EXPIRED, you set it to Disable. That is, even if the license is expired your application will continue running.

To know if the license is expired or not, you need to call the function WLRegGetStatus 161. Once that you know that it's expired, you disable updates in your application.

In order to renew the upgrade period again for another 12 months, you just need to send a new license (with 12 months expiration) to your customer.

1.8.8.3.21 When I protect my application, I keep on getting "your reg file is corrupted" even though it seems to have created the file properly. I have also installed the reg file correctly.

The base of the WinLicense licenses is the “License Hash” (in the Software 363 panel). Each software has an unique “License Hash” (generated in the Software Panel).

When you generate a license in the “License Manager” in WinLicense, you have to make sure that you select your specific software, so, the license will work for your protected application.

If you are using your own license generator, you have to make sure that you are generating the license with the expected "License Hash".

1.8.8.3.22 I need to change the license location and file name for each of my customers, but WinLicense seems to expect a fixed location and filename. What can I do?

You can change the file location and file name in runtime. To do that, you can make use of the Plugin system in WinLicense. Please, follow the next steps:

1) Create a simple DLL (which will be included in the Plugin System 52)

2) Create a function named **SecureEngineInitialize** (and export it) in your DLL. That function will set up the following environment variables:

WLDefaultLicenseFolder
WLDefaultLicenseName

WLDefaultLicenseFolder will be the location where you expect your license to be located
WLDefaultLicenseName will be the license name that you expect

3) Drag and drop the compiled DLL into the Plugins panel in WinLicense.

1.8.8.3.23 If I have already created a key which gives a user 30 days registered license, how do I go about giving another 30 days?

Every license has a different internal ID, so you just need to create a new license for that user with 30 days expiration. You don't need to change the user information (Name/Company/Custom Data...)

When WinLicense sees a license with a new ID, the expiration counters for that license starts from the beginning. That is, WinLicense stores the current number of days separately for each license.

1.8.8.3.24 Under Vista with UAC enabled, I call WLRegNormalKeyInstallToFile but the file license is created in a Virtual Folder in Vista. Is that correct?

That's a feature in Vista when UAC is enabled and the application is not running with admin's rights. The license file will be created in a Virtual Folder like:

C:\Users\UserName\AppData\Local\VirtualStore\Program Files\YourAppName

When WinLicense is reading the license key, Vista will redirect the file reading to that Virtual Folder (and if the file is not there, Vista will try to read the license from the same folder as your application).

To avoid those issues under Vista, you have the following options:

1) Create your file key under a defined folder in WinLicense. Please, check the following KB entry 332.

2) Use Registry licenses instead (with Registry root HKEY_CURRENT_USER)

3) Protect your application with a manifest to run your protected application with admin's rights. You can do this in the Extra Options 65 panel.

1.8.8.3.25 I use one project to protect three (*.exe) files. So I would like to name license key files like MyApp1.license, MyApp2.license, MyApp3.license. Is it possible with WinLicense?

Yes, you have to use the "%APP_NAME%" constant in the "File License" field in the Registration panel. Examples:

```
%APP_NAME%.exe.license

%APP_NAME%.license

%APP_NAME%.dat
```

1.8.8.3.26 What is the maximun length for Name, Company and Custom data in a license? I try to put quite a lot of data in the Name field but protected application fails.

In current versions of WinLicense, the Name and Company fields are limited to 256 bytes. Custom Data field can hold up to 8000 bytes.

1.8.8.3.27 What is the max. length of textkeys? Is this a fixed length for all generated textkeys or can it change?

The size if variable. It depends on the user information that you insert (Name, Company, Custom Data)

1.8.8.3.28 My application accepts both File and Registry licenses, which one has priority in case that a both of them are present?

By design, File licenses has priority over Registry licenses. In case that you allow both type of licenses in your application, please, make sure that your customers are not getting licensing problems due to mixture of both of type of licenses (due to license priority design).

1.8.8.3.29 I'm experiencing some problems using the WLRegDisableKeyInCurrentInstance function. How should this work?

If you call WLRegDisableKeyInCurrentInstance 148, the key will be ignored the next time that you restart your application. The application will go into trial mode and the application status will be "License disabled in current instance" (return value 15 when calling WLRegGetStatus 161).

If you go to the Customized Dialog 43 panel, you will see the message MSG_ID_LICENSE_DISABLED_ON_INSTANCE. That's the message that will be displayed by WinLicense when your application restarts, at least that you select that message and set it to

Disable (by right-clicking on it), so, your application will be launched normally and running in trial mode.

1.8.8.3.30 Everytime that some customers try to activate one of our product with the activation key (SmartKey), the program seems not to recognize it and the product still shows trial version. We use your WinLicense with hardware locked SmartKey's

It looks that you are handling most of the Customized Dialogs (in Customized Dialog 43 panel) with the option Disabled, so, all license errors are passed to your application (instead of Winlicense displaying an error message and close the application).

If you have set to Disabled most of the customized dialogs, please, make sure that you call the function WLRegGetStatus 161 to know the reason why the license is not registering (so, your application keeps running in Trial mode)

Notice that when you handle a message in the Customized Dialog panel (by setting it to Disabled), WinLicense will ***always*** keep your application running. That is, if for example you have set the MSG_ID_LICENSE_WRONG_HW_ID as disabled, WinLicense will keep your application running normally even if there is a license installed for another machine! (for a different hardware ID) You have to call the function WLRegGetStatus and, in this example, you should get the return value "wlInvalidHardwareLicense".

Also check if you have inserted a hardware ID in your generated SmartKey and your customer is installing the SmartKey in the expected machine (which matches the hardware ID in the license)

1.8.8.3.31 How can I generate my own license generators for my application protected with WinLicense?

You have multiple ways to create your own license generator:

1) Use the WinLicenseSDK.dll and call the license generator functions (please, refer to the help file for extended information about each function)

2) Use the CustomWinLicenseSDK.dll that is generated when you protect your application (when you enable the "Export Specific Generators" option)

3) Use our C ANSI code that you can compile to run your generator under any platform (Windows, UNIX, Linux, Mac.)

In the WinLicense examples folder, you have examples to create your own license generator via the WinLicenseSDK.dll

1.8.8.3.32 I generated a license for my software, now software is registered and WLRegFirstRun returns "True". Now I generate the same key again and put this license key into the software folder and WLRegFirstRun returns "True" again, is that correct?

When you re-generate a license (even if it has the same user information), WinLicense will create a new internal ID for the license. So, it's in fact a different license which flags the first time registration event for that new license. So, WLRegFirstRun 153 should return TRUE when a new license is present.

1.8.8.3.33 We have several customers who are going to trial state after changing from being online to offline with their laptop. What do you think could be the cause of this please?

If an application was registered and it changed into trial state suddenly it must be due to any of the following possibilities:

1) If you are registering via a file key, your customer (or any cleaning tool) has deleted the license file from disk, so WinLicense does not find any license file to register your application.

2) If you are registering via a Registry key, your customer (or any cleaning tool) has deleted the registry key from the Windows Registry, so WinLicense does not find any Registry key to register your application.

3) If you are handling most of the customized dialog messages related with registration (in the Customized Dialog 43 panel), you are probably not checking all possible license errors. Notice that if for example you are handling the MSG_ID_LICENSE_DAYS_EXPIRED (setting it as Disabled) and the license has expired by days, WinLicense will keep your application running even if the license has expired by days, because you are the responsible for checking if the license is expired (calling the WinLicense SDK functions). The same applies for any other Customized Dialog message that you set it as Disabled. You should call the function WLRegGetStatus 161 to get information about each possible licensing error and know why the application went into Trial mode.

1.8.8.3.34 I am getting quite a few of my clients telling me that they are getting expired key messages. They tell me that they have not changed their system. What can I do to help prevent this? And what might be causing it?

A license can expire “without reasons" when you have put expiration information in a license (like days or date) and your customer turns the clock back.

Sometimes the customer does not turn the clock back, but it’s Windows the one that synchronizes the clock (turning it back) and WinLicense detects that clock change. To avoid that, you have to go to the Trial Settings 32 panel and set the option "Clock Change" to "ALLOW_ONE_HOUR_BACK" or "ALLOW_ONE_DAY_BACK"

1.8.8.3.35 I am creating licenses with my own key generator (calling WLGenLicenseFileKey), but any license that I create it says "License corrupted" when I run my protected application.

Please, make sure that you are inserting the exact "License Hash" (the one that appears in the Software 363 panel for your application) in your license generator. If you insert a different "License Hash" in your code to generate the license, it will create a corrupted license for your specific software. The "License Hash" is the most important thing to generate licenses for your specific application.

1.8.8.3.36 Now I want to use "File License" for registration. In my application I ask the customer to select the registration file he received. But which command should I use to activate my application by this file?

To license an application via a license file, you just need to put the license file in the same folder as your application to protect. If you go to the Registration 36 panel in WinLicense, you can put the desired license file name that you want. Notice that you can also put a special directory where the license file will be located.

Please, refer to the following KB entry 332.

1.8.8.3.37 I want to license my application via Text keys, but I'm not sure how to proceed. Any help?

If you plan to work with Text keys, you need to have a place in your application to insert the text key (like an Edit box where your customer copies the text key). Your customer has to insert the text key in one of your applications dialog and you have to call the WinLicense SDK API to check and install that text key (as a file or registry key)

To check the text key that your customer has inserted, you have to call the API WLRegNormalKeyCheck 169. If the license is checked OK (WLRegNormalKeyInstallToFile returns TRUE), you have to install the text key as file or registry key (depending if you have enabled "File" or "Registry" keys in the Registration 36 panel in WinLicense). In case that you want to install the text key as a file key, you have to call the API WLRegNormalKeyInstallToFile 172. After that, your customer needs to restart your application (or call the API WLRestartApplication 258) and the application will start as registered.

1.8.8.3.38 Please, can you give me a step by step guide about how to create licenses locked to my customers' PCs?

1) Load your application in WinLicense and select the desired protection options.

2) Go to the Registration 36 panel and enable the option “This application can be Registered...”. We will leave just “File License” option for this test. You can also select the option “Application only runs when registered” if you want that your application ONLY runs when a license key is present (that is, it will not run in Trial mode)

3) Go to the Hardware lock 40 panel and select the items which will be checked from your protected application when a license with Hardware ID is present. We recommend you to uncheck the option "MAC" to avoid problems with special Wifi cards that change the MAC address from time to time

4) Protect your application

5) Now if you run your application and you enabled the option "Application only runs when registered", you will see that a message requiring a license is displayed and your application will not run. You can change the text for that message in the Customized Dialogs 43 panel (MSG_ID_LICENSE_REQUIRED_RUN)

6) Now, you can create Hardware locked licenses for your application. So, your application will *only* run in those computers where you create a license for them. Of course, you can create as many Hardware locked licenses as you desired without protecting your application again.

You can create licenses from the License Manager 391 (top button bar in WinLicense) or using the WinLicenseSDK.dll. For this example, we will use the "License Manager". So, open the License Manager and make sure that you have your Software created in the database and also the Customer that you will assign the license. From the Orders 374 panel, you can create a new License, please, set the "Hardware Id" for your customer's PC, so you lock the license to be run only in that specific computer (Read below how to obtain the Hardware ID from your customer). Once you have inserted all the information, press "Save" and you will have generated a license in the database.

You can send the license via email, using the Email Manager 379 in the License Manager.

7) Your application will run with the generated license key for the specific computer that you selected in the HardwareID field when you were creating the license in step 6)

How to obtain the Hardware ID from your customer

You can create a simple application which calls the function WLHardwareGetId 248. You can see an example of calling that function in the directory where you uncompressed WinLicense (WinLicenseSDK -> ExamplesSDK -> Trial Registration).

If you run your *unprotected* application which calls WLHardwareGetId, it will return a fake HardwareID taken from the file WinLicenseSDK.ini. When you protect your application, it will return a valid Hardware ID for the current computer. That's the HardwareID that you have to put when creating the license for your customer.

1.8.8.3.39 I have created text keys for 5 execution times. Now after 5 execution, I want to extent that execution time without protecting the application again. Is there any API for that?

Notice that if you create a license with expiration, like 5 executions, and it expires, the license is marked as expired and it will not work anymore.

You have to create a new license with new expiration in order to continue using your application. Of course, you don't have to protect your application again.

1.8.8.3.40 I am creating keys using "WLGenLicenseTextKey" API without using the WinLicense itself. But while using this API, it is asking one parameter called "LicenseHash". What shall I give for this parameter?

The "License Hash" is a unique number (long string) which allows creating licenses for your application. You can find the LicenseHash in the Software 363 Panel in WinLicense. Just press the "Software" button in the WinLicense toolbar menu and select your software that you are going to protect. Go to the License Hash field and you can copy the string into formatted Delphi or C/C++ by right-clicking on it.

In the source code of your license generator, you have to pass the License Hash to any of the functions to generate licenses for your application.

In the WinLicense folder, you can find examples to generate licenses for your application (%WinLicense folder%/WinLicenseSDK/ExamplesSDK/Generators)

1.8.8.3.41 Can I only use locked keys to register my application?

Yes. To create locked license keys, you have to specify the machine ID before creating a new license key (License Manager). If you want to make sure that your application will only accept machine locked license keys, you can force WinLicense to check for them. To do so you have to go to the Registration panel and check the option Allow only hardware dependent (locked) registrations.

Please, go to the following KB entry 338.

1.8.8.3.42 How can I get extended information about the current registration status for my application?

To do this, the WLGetRegStatus 161 WinLicense function can be used. The return value will tell you whether the license is correct, expired or if the current license key does not have machine ID information, etc.

1.8.8.3.43 I have received a report about a stolen license key. How can I block it?

Yes, you can do it with WinLicense. Please, refer to the following section 97.

1.8.8.3.44 Is it possible to implement a feature that prohibits my application from being run with a permanent key? In my application all keys must expire.

To force WinLicense to always check for an expirable license you have to check the "Accept only temporary keys (that expire)" option in the Registration 36 panel. WinLicense will reject license keys that have an unlimited usage period.

1.8.8.3.45 I can see several kinds of licenses to register my application (Single file, registry, text file..). Which one should I use to make my application more robust against cracking?

WinLicense offers different types of licenses to satisfy different developers' needs. In all types of licenses, WinLicense offers a strong encryption algorithm to protect a license from being tampered with or faked. You can find more information here 93.

The examples below show how each type of license looks:

Registry license key

Registry keys are very easy to install in the system, without requiring computer knowledge for your customers and it's totally transparent for your application. To install a registry license, your customer has to double-click on the .reg file (created by WinLicense) and accepts the standard Windows message box to install the key in the Windows registry. After that, your application needs to be restarted in order to finish the registration process for your application.

Example of registry license (*.reg file):

```
REGEDIT4

[HKEY_LOCAL_MACHINE\Software\Company\Product]

"reg_key"=hex:38,4A,0F,10,C0,DB,C1,5C,04,82,31,7C,8E,CC,5B,2E,78,39,74,7F,5E,DB,77,C7,DA,39,B4,DD,F1,1E,10,70,1F,98,45,79,

4F,57,94,03,96,A8,98,27,B1,9A,6A,B5,9E,31,29,6E,8F,F7,DD,A6,CD,11,88,43,A1,50,28,14,89,0F,D1,BD,88,D4,A9,90,97,8B,77,00,00,80,

3F,9F,4F,27,13,DC,73,75,3F,E8,B7,A1,1F,CC,19,B6,9D,8A,D2,1D,C7,A7,8D,98,95,8D,93,7A,71,7B,00,80,3F,9F,4F,27,13,89,96,71,D5,

CD,DE,62,02,BC,1A,B6,B9,88,0F,3D,61,30,17,0B,05,02,01,00,FE,FF,FF,FF,FE,FF,FF,FF,EF,30,AB,20,D7,BB,98,08,97,2F,0F,CB,08,AF,
```

11,A2,CC,32,57,9F,B1,49,0D,19,55,06,4C,FE,09,67,6F,AD,7D,C5,7A,FE,3C,F0,5E,47,67,93,47,56,D
C,32,3D,2D,08,CE,D5,35,C8,41,4C,

F8,39,C1,4E,CF,FD,44,94,CC,E2,5B,4A,46,86,18,89,2B,3A,79,AC,DA,AE,D7,B7,2B,6D,02,9C,74,98,
A5,84,83,0D,45,7A,5A,39,E0,12,63,

F9,53,89,25,6A,D3,8B,FC,2E,57,D1,F9,13,6E,87,73,B7,2A,C6,58,6B,8B,E9,07,DF,E9,F4,58,9E,14,D9
,A1,C9,B7,C1,B0,3E,57,B7,87,6A,

F2,4F,90,2A,66,C6,52,9B,E5,C8,29,5F,69,0E,27,44,80,C4,A0,E8,3C,03,86,9C,9D,26,35,10,FC,31,86,
CF,26,16,CF,FA,C9,FE,DD,6F,69,

F4,B4,9B,04,8D,BD,5B,78,03,80,CC,F7,05,57,90,7B,4B,54,75,92,01,CE,19,4F,40,B3,CD,AF,63,62,4
1,0E,6F,B3,00,39,53,FC,2B,DC,3B,

0B,A0,7B,31,E2,CC,16,CA,EA,8C,8A,40,AD,FD,09,43,84,C1,8D,88,81,A6,A4,3E,FB

Text license Key

Text keys are convenient when you want to handle the registration of your application with a specific form inside your application. Text keys can be easily sent by email to be copy/paste into your specific registration form.

Example of text license key:

6DEA68EC78BD5B357EC6A4E8A91D018911A6B3F0691142B5EF65762145C7ADF7549B3F7FE
39829E605F795C6C573BD90D928

2F6E8FF7DDA1417C3E9ECFE7F3F9FC084E6C0074777F900852602F170B0502803FCCE033944
05CECC5295755686FC51644660

5A88B8494F950E92F170B0502803F1F3D377E9BB537EC395855846D82F6BEDF6F371B0D068
3FEFFFFFFFEFFFFFFC3FC53DC913

F1D4B80AF3DDDBD297C5114849630F716E28EDA20CC81A89E5314060B0824DDC29347D3
86AD0F6DEB1703AB4E344E9ABE54E

0D73893542E93AD331126F991F42FE384C2AD6A17201A1F27F7D1AA4AED95C41287FA2E06
C55D4B51B4CD6BE3F147AA5748A2C

4362B3510950E3FFA87DCBC811A3A29362A11E1C14D07A5DB15A1094609DF6C6254CEDC8
2E60B57C15A62B1DB394544279828

4E118BF6CB981D54384D2D2BF0D85021B4F218BB185D0CF97B7957E8EB99E92566D85D7C
C0F23C5F533E9B425D288E10DBAF

206E47643045FFEF533BC3091CD527740F138B905A02FBB0EC3F75EF6096CF09407962559E

File license Key

File keys are special files that hold the licensing information for a specific application. To register your application with a file key, your customer has to put the file key in the same folder as your application. After that, your application needs to be restarted in order to finish the registration process for your application.

SmartActivate® license key

The SmartActivate® System allows an application to be licensed using a valid activation code that should be inserted in a dedicated form in your application. It is very popular now to use such keys when almost all registration data comes as plaintext and the key has a checksum for data and license restrictions.

Example of SmartActivate® license key:

Name: John Smith

Company: J.S. Software Development

Custom Data: MODE: Advanced

Key: 286D0394-87716B52-BC59FA9A-CAA8DE87-4881A155-BF450169-C590

1.8.8.3.46 I have 2 programs but I want each to have a specific license key. How can I do that?

To do this, you have to use different license hashes for each software application. This can be done with the help of the Software Manager 363 panel. After changing your License Hash, your application needs to be protected again, so it will accept the licenses for the new License Hash.

1.8.8.3.47 How can I use a single license to register all my products?

To do this, you must use the same "License Hash" (from the Software 363 panel). The License Hash is a big number which represents the seed to generate licenses for a specific Software. If two software have a different License Hash, the licenses generated for one software will not be valid for the other software.

To copy/paste the License Hash (to share the same license between different software), just right-click on the License Hash edit box and select Copy/Paste from clipboard.

1.8.8.3.48 I don't want my application to be executed unless a valid license key is present. How can I do that?

This can be done using the "Application only runs when registered" option (in the Registration 36 panel).

If there is not a license and you run the protected application, WinLicense will display the customized dialog MSG_ID_LICENSE_REQUIRED_RUN (in the Customized Dialog 43 panel)

1.8.8.3.49 Is it possible to lock my application to a specific machine?

Yes. The Hardware Lock feature allows this type of restrictions. No one can use a hardware-locked license key on a computer that has a different hardware configuration.

Please, refer to the following KB entry 338.

1.8.8.3.50 Can WinLicense protect stand alone DLLs that I would sell? Customers would just buy my DLL to integrate it with their own complete application.

You can protect DLLs normally as you do with applications (EXEs). You can put Trial 32 or Registration 36 options in your DLL.

You might want to protect your DLL to be executed *only* when a valid license is present in the directoy where your DLL is located. So, you have to activate Registration options before protecting your DLL (and mark the option "Application only runs when registered" if you want that your DLL only executes when it's registered with a valid license key).

You can also protect your DLL with Trial options, so, your DLL can run in Trial mode for a specific period or with options disabled.

1.8.8.3.51 Is there a way where I can clear of the registration information and run the same application in trial mode?

You can use the SDK function WLRegRemoveCurrentKey 176 or just explicitly deleting the file license (or registry key) which holds the registration information. Basically, that function removes the file from disk (or the Registry license from the Windows Registry).

If you want to fully invalidate the current license, so it cannot be used again, you have to call the function WLRegDisableCurrentKey 147.

1.8.8.3.52 How do I check to see if a license has a resistration date set? When I use the function WLRegExpirationDate(addr(TrialDate)) with no expiration date set then I receive an error "invalid argument to encode date"

If you don’t insert Expiration date in the license, the return value for WLRegExpirationDate 151 is -1

You should not display the date if you get -1 as the SYSTEMTIME structure is filled with 0,0,0

If you are generating license keys with different expiration restrictions (by days, executions, date...) You can first call the function WLRegGetLicenseRestrictions 158 to know the restrictions in the current license.

1.8.8.3.53 I also miss the option in armadillo to insert in the serial number from the license generator an integer value that could be extracted in the program. I used this to insert the number of licenses the user registered into the serial number.

You can use the Custom Data field when generating your license. So, you specify the information that you want (you have 8000 bytes to put any information that you want).

1.8.8.3.54 What is the best way to register the program for ALL users on the computer?

In the "File License" field in the Registration 36 panel, you can use a special constant that refers to a folder common to all users. For example, you can use the following constants: %COMMON_APP_DATA%, %PUBLIC_DOCS%

You can use sub-folders when using the above constants.

Example:

```
%COMMON_APP_DATA%\Myapp\Licenses\license.dat
```

1.8.8.3.55 We have problems with a few customers. We use SmartKeys with hardware id, the SmartKey is verifies correctly, but after relaunching it, they see that the program is still unregistered.

If you are using SmartKeys, you have to call WLRegSmartKeyInstallToFile 182 or WLRegSmartKeyInstallToRegistry 189, as you might know.

For restricted users or admin in Vista with UAC enabled, they cannot write to Registry root HKEY_LOCAL_MACHINE and for files, they might not have privileges for writing to the current directory.

So, we recommend you to change the place where your license will be stored in the Registry or file. So, you have to go to the Registration 36 panel and if you are going to install the SmartKey into the Registry, please, use "HKEY_CURRENT_USER".

If you are going to install the SmartKey into file, then you have to change the "File License" name into one of the new defined constants. Please, refer to following KB entry 332.

1.8.8.3.56 Our users must be able to generate keys from a website. Is that possible via PHP?

We don't have PHP code to generate licences, but we can provide you with C ANSI code that you can compile create licenses in non Windows platforms (like Linux, Unix, etc). You can create a PHP front-end which just calls your compiled key generator. This source code is available for any of our customers who request it.

For Windows platforms, you just need to call the WinLicenseSDK.dll (API generator functions 194), so you don't need to compile the key generator.

1.8.8.3.57 I have a single Delphi application that has 12 modules, depending on what the Customer needs/wants. They may start off with the basics then add others as their needs grow. How would WinLicense facilitate that?

Please, notice that in the license fields, you have “Name”, “Company”, “Custom Data”, etc. The “Custom Data” is commonly used to put specific information for that license. So, you can enter specific values in the “Custom Data” to change the behavior of your application and restrict or allow functionality inside your application.

You just need to call the API “WLRegGetLicenseInfo 156” and you will get the “Custom Data” field inside your license, so you can restrict or allow operations in your application depending on the information that you get for the “Custom Data” in the current license.

1.8.8.3.58 I'm not sure how to work with the License Manager in WinLicense 3.0, can you give me the basic steps?

1) Create a new Software (from the top menu "Software" button)

2) Load your Software in the "Application Information" main panel in WinLicense

3) In the Registration panel, check "Registration" and select for example "File License --> regkey.dat"

4) If you want that your application only runs when registered, check the option "Application only runs when registered" (in the Registration panel)

5) Click on Protect

Now when you run your application, you can see that it displays the message "License required to run" (you can edit it in the "Customized Dialog" panel)

Now, you can create a test license for your application:

1) Go to the License Manager

2) Add a Customer (from the "Add" customer top button)

3) Click on "Add Order" and select there your Software, your Customer, and go to the "License" tab. Make sure that "File Key" is selected as that's the type of license that you want to generate (as you put in the "Registration" panel). Click on Generate and you can change/set the license details. After that press OK.

4) You can see that the license appears in the "Registration content" panel. You can right-click on that panel and select "Export Binary License" from the popup menu. Save the license as "regkey.dat" (as that's the name that you put in the "Registration" panel) and put it in the same folder as your protected application. You can change the default license name and location from the "Software Custom Values" (in the Software panel)

5) Run your protected application again and it should run as registered.

Notice that you have the order created in the database, it should appear in the "License Manager" main screen.
You can email the license to your customer by creating an email template and attach the license. Please, refer to the top menu "Email Settings --> Template Manager"

1.8.8.3.59 In version 2.x, I was generating licenses with the WinLicenseSDK.dll. Do I have to change it for version 3.x?

Version 3.0 has a new format for the "License Unique key" (License Hash). If you have migrated your old License hash (from version 2.x) to the new format using the "ConvertDatabase.exe" tool, then you can continue using your old license generator with the old WinLicenseSDK.dll

If you have created a new license hash in version 3.x, then you need to use the new WinLicenseSDK.dll that is shipped with WinLicense 3.x

Basically, the WinlicenseSDK.dll that is shipped with version 2.x, can create licenses for the old "License Hash" format. The new WinLicenseSDK.dll that is shipped with version 3.x, can create licenses for the new "License Hash" format.

1.8.8.3.60 I'm using the C ANSI generator to generate licenses for my application protected with version 2.x. Do I need to change the C ANSI source code to generate licenses for applications protected with version 3.x?

The old C ANSI generator can generate licenses for the old "License Hash" format.

Version 3.x has a new "License Hash" format. In case that you want to generate licenses using the new "License Hash" format, you have to update your C ANSI generator. Please, contact us at support@oreans.com for more information.

1.8.8.3.61 When I call WLRegNormalKeyCheck, it always report TRUE even if I don't enter any information. What's wrong?

Notice that when you run the application in unprotected state, you are just taking the *fake* values from the WinLicenseSDK.ini file. In that file, you have set that WLRegNormalKeyCheck will return "TRUE".

You have to protect the application in order to get the SDK working as expected. The WinlicenseSDK.ini is only used to allow you testing your application with the SDK while it's not protected, so you can debug it easily.

1.8.8.3.62 How can I convert my winlicense.abs (from version 2.x) to version 3.x?

To convert the database from version 2.x to version 3.x, please, follow the next steps:

1) Open the application "ConvertDatabase.exe"

2) Drag and drop your winlicense.abs file into the ConvertDatabase main window and select where you want to export the database (as XML files)

3) Open WinLicense 3.x (WinLicense.exe) and go to the toolbar button "License Manager"

4) In the License Manager, click on the "Database" tab and select "Import from XML"

5) Select the folder where you exported the XML files (from step 2)) and you are done

You can go to the "WinLicense" main panel and click on "Load Project" to load any of your converted projects. In the License Manager in version 3.0, you can see all your previous generated licenses and customers.

NOTES:

1) You should review your exported projects as not all the options match with the new version.

2) If you are using "WL DLL Control", you have to change your DLL to use the new Plugin System 52.

3) If you are using "SecureEngine Config" options, please, contact us for equivalent options.

1.8.8.3.63 How can I export specific information from the database to XML?

You can use SQL to query the WinLicense database to display and export any specific information from the database.

From the License Manager, click on the "Database" group and select "Manage". In the "SQL Query" text box, you can execute any specific query. For example, if you want to display all the orders that are stored in the database, you can execute something like:

```
SELECT ID_Order, Product_Name, Customer_FirstName, Customer_LastName
FROM
  Orders, Customers, Products
WHERE
  Orders.ID_Customer = Customers.ID_Customer AND
  Orders.ID_Product = Products.ID_Product
```

The displayed data can be exported as XML or CSV.

1.8.8.3.64 In version 2.x, I saw an Activation option and it looks that's what I need now, but I can't see that option in version 3.0

In latest versions 2.x, we included the Activation system but it was commonly used by our customers, basically it was limited to be used with our (freeware) application "WL Orders Manager" and that was quite rigid. The Activation system allows to send Activation codes to your client.

With the Activation Code, the protected application connects to your server (where you have the "WL Orders Manage" database (MySQL)) and a license is created there (using the shipped compiled license generators). Once the license is created, it's sent to the customer's machine and the application runs as registered. It also offered the chance to migrate the license to a different Hardware ID.

In version 3.0, we have removed the Activation system (at least from our User Interface), but it can still be used with some internal options (read below)

Anyway, not sure if you are ready to work with WinLicense Activation and our product "WL Orders Manager". It requires some (minor) learning process to integrate everything together. In the Help File in version 2.x and the Help File in "WL Orders Manager" you have information about how to make it work the Activation in WinLicense.

The system works more or less like this:

1) You receive an order/payment from a customer

2) You insert that order information in "WL Orders Manager" (User name, company, license restrictions if required, etc). After that, "WL Orders Manager" will generate an Activation Code that you can send via email (you can send it from WL Orders Manager). You can also set up specific Activation restrictions (like if re-activation is allowed, max computers to activate, etc)

3) Your customers inserts that activation code in a form (that is injected into your protected application) and it will contact your website (which will hold a simple PHP script that we will send you) that checks the activation code and create the final WinLicense license key (locked to your customer's machine hardware ID) and send it back to the protected application, so it will run as registered.

4) Now, from "WL Orders Manager" you can see the Activation status and other information, like, when each customer activated his license, if a customer has re-activated the license, how many computers are activated with a specific "Activation code". You can also disable a specific Activation code, so it cannot be reused again, etc.

5) Your customer can revoke his current license (locked to his machine) so he can apply the activation process again on a new machine. This allows your customer to move his license to a different PC. Of course, you can control how many times your customer can perform this process, etc.

Basically, your webserver will hold the "WL Orders Manager" database (which is a MySQL database) and the PHP script that will access the database and generate licenses. The "WL Orders Manager" is just a Windows application that can access the MySQL database in your webserver.

Options to enable Activation in WinLicense 3.0

You can add the following lines in the Advanced Options 67 panel. Just edit the Host Name and activation/deactivation URLs as you had in the Winlicense GUI in the “Activation” panel for version 2.x:

```
OPTION_ACTIVATION_IS_ENABLED=YES
OPTION_ACTIVATION_HOST_NAME=127.0.0.1:80
OPTION_ACTIVATION_ACTIVATION_URL=/activate.php
OPTION_ACTIVATION_DEACTIVATION_URL=/revoke.php
```

If you were using an external DLL for the activation process, then you also have to add the following lines (changing the PATH to your DLL and the Function Handler name that is exported from your DLL). Something like this:

```
OPTION_ACTIVATION_IS_USE_DLL=YES
OPTION_ACTIVATION_DLL_NAME=C:\MyPath\ActivationDLL.dll
OPTION_ACTIVATION_FUNCTION_HANDLER=ActivationHandler
```

1.8.8.3.65 I need to use a very long custom data for my file licenses. It can take about 100Kb of custom data but WinLicense rejects such long licenses. What can I do?

For file licenses, we are now allowing any size for the custom data. Notice that it's only available for licenses created with the function WLGenLicenseFileKey 195/WLGenLicenseFileKeyEx 195, WLGenLicenseRegistryKey 203/WLGenLicenseRegistryKeyEx 207 (and their unicode versions)

When you protect your application, there is a special option where you can set maximum size for the custom data. Notice that by default WinLicense accepts up to 8000 chars for the custom data. In case that you want to increase that size, go to the Advanced Options 67 pane and add the following line:

```
OPTION_ADVANCED_MAX_CUSTOM_DATA_SIZE=100000
```

In the above example, we have put "100.000" bytes for the custom data. Notice that for UNICODE licenses, you have to specify the double of bytes. For example, if you are creating a UNICODE license with 10000 chars, you have to specify the value 10000 * 2 = 20000 in the above option.

1.8.9 Hardware Lock

- Is it possible to lock an application to multiple machines with Hardware Lock? 353
- How do I check if a user has entered a valid hardware ID without any typos? 353
- I want to display the hardware ID when the option "Application only runs when registered" is used. 353

- What about the option Ring-0 in version 3.0? I was using it to create licenses with Ring-0 hardware ID 358
- Is the hardware ID the same with WinLicense 2.x and WinLicense 3.x? I have many hardware locked licenses created for my application protected with version 2.x 358
- Can I send several Hardware IDs in a single license to be used by several computers? 358
- In version 2.x when I put %machineid in my Customized Dialogs, the ID was autmatically copied to the clipboard. This does not work on version 3.x

1.8.9.1 Is it possible to lock an application to multiple machines with Hardware Lock?

The Hardware Lock feature allows you to lock a specific license to a single machine. There is not a direct way put multiple hardware IDs on a single license. You could do the following workaround:

1) Get the Hardware ID of each machine that you want (calling WLHardwareGetId 248)

2) Create a new license and in the "Custom Data" field you insert a string containing all your Hardware IDs (for all your wanted machines)

3) In runtime, you call "WLRegGetLicenseInfo 156" and you read the "Custom Data" field, so you compare the current Hardware ID (via WLHardwareGetId) with all the hardware IDs that are embedded in the "Custom Data" field. So, you allow execution or not.

1.8.9.2 How do I check if a user has entered a valid hardware ID without any typos?

You can just use the function WLHardwareCheckID 246. That function returns if a Hardware ID is valid or not.

1.8.9.3 I want to display the hardware ID when the option Application only runs when registered is used.

If you check the option "Application only runs when registered", WinLicense won't start your application at least that a license is present. If you just want to display the Hardware ID, you can just go to the "Customized Dialog" panel and edit the MSG_ID_LICENSE_REQUIRED_RUN and insert the string "%machineid" anywhere within your message, so it will display the current Hardware ID when the message appears asking for a license appears. Example for the customized message:

```
"Sorry, this software requires a key to run. Your current Hardware ID is
%machineid"
```

1.8.9.4 How can I enable hardware lock in smartkey protection? Can you give me some examples?

You can create Hardware locked smartkeys with WinLicense easily. For example, you just need to go to the "License Manager" panel and add a new license inserting Hardware ID before generating the license. The generated license can be only installed in a computer with the Hardware ID that you inserted.

1.8.9.5 If I set BIOS changes to 0 in the HardwareLock panel, does it mean that the User can change the BIOS unlimited times or the User cannot change BIOS?

BIOS Changes to 0 means that no Hardware changes are allowed. As soon as your customer changes the BIOS in his computer, the license will not work anymore.

1.8.9.6 How can I get the Hardware ID of a specific machine to create a hardware locked license?

Please, refer to the following KB entry 338.

1.8.9.7 I have seen that WinLicense can lock a specific license to a USB device, does it mean that my customer only needs to move his USB device to different computers to keep using the protected application with his hardware locked license?

Exactly! USB devices can be used as common dongles in WinLicense. USB locked licenses will give total freedom to your customers to move across computers when you use hardware locked licenses.

1.8.9.8 I have protected my application with the following "Hardware Lock" items: CPU + BIOS, but now I want that my new protected application will be locked to CPU + BIOS + HDD. Do I have to obtain again the Hardware ID of my clients?

No, you just need to get once the Hardware ID of your clients. After that, you can change the Hardware Lock items with no problems.

Notice that the hardware ID retrieved by WLHardwareGetId 248 is the full hardware ID. WinLicense will just skip from checking the fields that you have unchecked in the Hardware Lock panel.

1.8.9.9 Is it possible to supply a custom hardware ID to your API? So, I can use my own hardware ID and not the one generated by WinLicense

Yes, you can change the internal hardware ID in WinLicense by your own one. To do so, just use the Plugin 52 system in WinLicense, specifically the function SecureEngineProcessHardwareId 56.

1.8.9.10 I have a issue where it appears some of my customers hardware ID's will change after they reboot. Do you have any idea what may cause this?

Can you check if you have enabled the option “MAC Address” in the Hardware Lock 40 panel?

Notice that some wireless devices change the MAC address when they reconnect and that might be your problem. You should uncheck the “MAC Address” option in the Hardware Lock panel and reprotect again your application. You don’t have to ask again for the HardwareID to your clients (WinLicense will just ignore the MAC Address from the HardwareID string and will work as normal)

1.8.9.11 If I select different setting in the "Hardware Lock" panel, I always get the same Hardware Id, how is that possible?

The hardware ID has an internal format and it will always display the same hardware ID for a specific computer, no matter if you check/uncheck different "hardware items" in the Hardware Lock 40 panel.

In runtime, WinLicense will ignore specific fields in the Hardware ID according to your settings in the Hardware Lock panel. This approach gives flexibility to change, at a later time, the Hardware Lock settings without having to ask again for a new hardware ID to your current customers.

1.8.9.12 What if my client flash-update the BIOS. Will this change the hardware ID?

WinLicense reads the manufacturer BIOS serial number. In common flash or BIOS updates, the serial number should not be altered, so the hardware ID should stay the same after a BIOS update.

1.8.9.13 Will repartition or reformat the hard drive change the hardware ID?

WinLicense uses different methods to read the HDD serial number, as some methods do not work in specific computers. The common scenario is that the serial number is the same no matter the method that is used to retrieve the HDD serial number. Unfortunately, for some specific computers we have seen that the HDD serial number changes when the Windows version has been upgraded on the same computer (for example, when migrating from Windows XP to Windows 10).

1.8.9.14 Is the CPU ID the same between different Windows versions?

Yes, the CPU ID should be the same across different Windows version on the same computer.

1.8.9.15 Is the CPU ID enough to be used as hardware ID, so I don't rely on HDD, BIOS, MAC for the Hardware ID?

The CPU ID is not enough to create a unique ID. The CPU ID is the same for a specific CPU model, for example, two computers with the same processor model (i.e: Intel Core i7-9900K) will report the same CPU ID.

1.8.9.16 What are the different ways to lock a license to a specific device?

WinLicense has 3 ways to lock a license to a specific machine ID:

- **PC Hardware**: You have to set this option in the Hardware Lock 40 panel. After that, you can retrieve the PC Hardware ID by calling the function WLHardwareGetId 248. Once that you have the hardware ID, just add it to the license when you generate it.

- **USB drive**: You have to set this option in the Hardware Lock 40 panel. Now you have to get the ID of the specific USB drive that your customer wants to use to lock the license to it. As the user might have several USB drives connected to his machine, you should display the ID of all different USB drives that are found on his machine.

  You have iterate over all USB drives and display their names and IDs, so the customer can recognize the specific USB drive that he plans to use for hardware locking. To accomplish that, WinLicense offers 3 functions WLHardwareGetNumberUsbDrives 250, WLHardwareGetUsbNameAt 254, WLHardwareGetUsbIdAt 251.

  The following C code shows how you can use the above 3 functions in order to display the ID of all found USB drives on a system:

```
int number_usb_drives = WLHardwareGetNumberUsbDrives();

for (int i = 0; i < number_usb_drives; i++)
{
    char usb_name[256];
    char usb_id[256];

    WLHardwareGetUsbNameAt(i, usb_name);
    WLHardwareGetUsbIdAt(i, usb_id);

    printf('%s: %s', usb_name, usb_id);
}
```

  The above code will output something like:

```
SandDisk Cruzer Edge USB Device = 4A32-8016-A611-43A1-9644-AAA4-7CB1-F22B
Thosiba USB Device = 5A32-E835-B14C-2E5A-B6FA-B21C-6A12-A31C
```

  Now the user will recognize which USB he will use for hardware locking and he will send you the specific USB ID. After that, you create a license locked to that specific ID

- **Custom ID**: You can compute your own hardware ID and put in a WinLicense license via the Plugins Manager, specifically, using the callback SecureEngineProcessHardwareId 56

1.8.9.17 How can I lock a license to my customer's USB pendrive?

In order to lock a license to a USB drive, please, follow the next steps:

1) In the Hardware Lock 40 panel, check the option "Activate USB (drives)"

2) Protect your application

3) Retrieve the USB hardware ID of your customer's pendrive. To do so, you can create a separate tool to retrieve the USB IDs or adding that functionality somewhere in your application.

You have iterate over all USB drives and display their names and IDs, so the customer can recognize the specific USB drive that he plans to use for hardware locking. To accomplish that, WinLicense offers 3 functions WLHardwareGetNumberUsbDrives 250, WLHardwareGetUsbNameAt 254, WLHardwareGetUsbIdAt 251.

The following C code shows how you can use the above 3 functions in order to display the ID of all found USB drives on a system:

```
int number_usb_drives = WLHardwareGetNumberUsbDrives();

for (int i = 0; i < number_usb_drives; i++)
{
    char usb_name[256];
    char usb_id[256];

    WLHardwareGetUsbNameAt(i, usb_name);
    WLHardwareGetUsbIdAt(i, usb_id);

    printf('%s: %s', usb_name, usb_id);
}
```

The above code will output something like:

```
SandDisk Cruzer Edge USB Device = 4A32-8016-A611-43A1-9644-AAA4-7CB1-F22B
Thosiba USB Device = 5A32-E835-B14C-2E5A-B6FA-B21C-6A12-A31C
```

Now the user will recognize his USB pendrive from the list and he will send you the specific ID.

4) After receiving the USB ID from your customer, just create a license locked to that specific ID and send the license to your customer.

1.8.9.18 When I call WLHardwareGetNumberUsbDrives() I always get zero. I'm sure that I have inserted a USB memory stick

You have to make sure that you have checked the option "Activate USB (drives)" in the Hardware Lock panel 40. If you don't check that option, WinLicense will not include the code to read USB drives into the protected application.

1.8.9.19 What about the option Ring-0 in version 3.0? I was using it to create licenses with Ring-0 hardware ID

In version 3.0 we have totally removed the Ring-0 option.

In version 2.x, we mostly removed the Ring-0 option and just left it to get the hardware ID for old Windows versions (from Windows 95 to Windows XP). In case that you have your licenses generated with the Hardware ID with the Ring-0 option, you have to retrieve again the Hardware ID for your Windows XP users.

In the Hardware Lock panel, make sure that you set "Hardware Lock Engine --> VERSION_1_0" (in the Hardware Lock 40 panel)

1.8.9.20 Is the hardware ID the same with WinLicense 2.x and WinLicense 3.x? I have many hardware locked licenses created for my application protected with version 2.x

In order to retrieve the same Hardware ID as in version 2.x, please, go to the Hardware Lock 40 panel and set the option "Hardware Lock Engine" to "VERSION_1_0"

1.8.9.21 Can I send several Hardware IDs in a single license to be used by several computers?

Unfortunately, you can only insert one Hardware ID per license. You have 2 options:

1) Use the new USB hardware locking. In that case, your customer can move the USB drive to a new computer and run your applicaiton there

2) You can use a workaround that might help you:

> 1) When you generate a new license, in the Custom Data field of the license, you could insert a list of all the Hardware IDs that you want to include.
>
> 2) Do not use the option "Hardware ID" in the license
>
> 3) Create the new license
>
> 4) Now, you can see that the license starts on *any* machine when running your protected application

5) In your source code, you should call the function "WLRegGetLicenseInfo 156" and you get there the list of "Hardware IDs" that you inserted in the "Custom Data" field

6) From your source code, you call the function WLHardwareGetId 248 and you can compare each of the previous hardware IDs (from the custom data) and see if it matches the one with "WLHardwareGetId". If so, you allow execution of your application. If not, your application refuses to run.

1.8.9.22 In version 2.x when I put %machineid in my Customized Dialogs, the ID was autmatically copied to the clipboard. This does not work on version 3.x

If you want to copy the hardware ID to the clipboard when it's displayed in any of your Customized Dialogs, please, go to the Advanced Options 67 panel and add the following entry:

```
OPTION_ADVANCED_COPY_HARDWARE_ID_TO_CLIPBOARD=YES
```

1.8.10 Sales

- Can you tell me about how WinLicense subscriptions work? 360
- I have paid via Shareit but you have not sent me any invoice. Can you send it, please? 360
- I have paid via Fastspring but you have not sent me any invoice. Can you send it, please? 360
- If we purchase your software via Bank wire transfer, will you provide our company with an invoice for our purchasing? 360
- Does initial purchase include some degree of update support, or must I purchase the update subscription at the outset to get updates during the first year (or whatever period)? 360
- I paid to Shareit via bank wire transfer, why I have not received your software yet? 360
- As soon as the free update period (12 months initially) expires, I will need to renew WinLicense subscription. Do I have to pay the renewal fee before it expires or is it possible to pay somehow later (if I happen to forget)? 361
- What is the difference between "Developer License" and "Company License"? 266

1.8.10.1 Can you tell me about how WinLicense subscriptions work?

When you purchase WinLicense, you will have 12 months of free upgrades and technical support. In order to keep our projects alive and up dated we require a small amount of capital to keep producing the updated versions. When your original subscription expires all you have to do is purchase a new subscription.

We have 2 types of subscriptions: 12 months subscription or 6 months subscription. Please, check Prices and subscriptions for more details (http://www.oreans.com/order.php)

1.8.10.2 I have paid via Shareit but you have not sent me any invoice. Can you send it, please?

Please, notice that Shareit changed the way that orders are processed. When you purchase a product from us, Shareit is the one that purchases the product from us and sells it to you (that is, you are purchasing the product to Shareit and not to us) So, Shareit is the one that emits a valid invoice for you.

1.8.10.3 I have paid via Fastspring but you have not sent me any invoice. Can you send it, please?

Please, notice that when you purchase a product from us, FastSpring is the one that purchases the product from us and sells it to you (that is, you are purchasing the product to FastSpring and not to us) So, FastSpring is the one that emits a valid invoice for you.

1.8.10.4 If we purchase your software via Bank wire transfer, will you provide our company with an invoice for our purchasing?

Sure, just let us know your company details that you want to appear in the invoice and VAT ID for European Union countries and we will send you a valid invoice for your purchasing (in PDF format)

1.8.10.5 Does initial purchase include some degree of update support, or must I purchase the update subscription at the outset to get updates during the first year (or whatever period)?

When you purchase any of our products, you have 12 months of free updates and technical support. After that period, you need to get a subscription plan to keep receiving updates and technical support. Of course, you can continue using our products even if your subscription period expires (though you won't be able to receive new updates of our software until you purchase a new subscription plan)

1.8.10.6 I paid to Shareit via bank wire transfer, why I have not received your software yet?

You have paid via bank wire transfer to Shareit. In that case, Shareit is the one that collects the money and send us the notification when everything is processed. So, you will receive your license as soon as we get the Shareit notification email.

This take about 4-5 business days to process by Shareit. You might want to contact Shareit for detailed information about the status of your transfer.

If after 5 days you have not receive any news from us, that means that Shareit has not made any notification to us about your payment. Please, let us know your Shareit order number and we will contact Shareit directly for further information about your oder.

1.8.10.7 As soon as the free update period (12 months initially) expires, I will need to renew WinLicense subscription. Do I have to pay the renewal fee before it expires or is it possible to pay somehow later (if I happen to forget)?

You can renew it at any time, even if your free update period expired for several months. When you purchase the subscription plan, it will start again the day that you purchase the subscription plan.

1.8.10.8 What is the difference between "Developer License" and "Company License"?

Company licenses are for companies with more than one software developer. All developers in the company can use the license to protect all the software developed within the company.

The single **Developer License** is intended to be used for single developers or companies with **just one software developer**. If there is more that one software developer, then you need to purchase a company license.

If you purchase the "Developer License", you can protect all your projects with your license. Notice that **you cannot protect projects which are not developed by you**.

The same is applied to "Company Licenses". That is, the company license can be used by any employee inside the company, but they can **only** protect software developed **inside** the company where the license belongs.

The company license is suitable for a company inside a region (country), but not for overseas offices.

1.9 Support

If you have any technical problems using WinLicense or need a special feature to be included in a next release, please feel free to contact us at support@oreans.com.

License Manager

2 License Manager

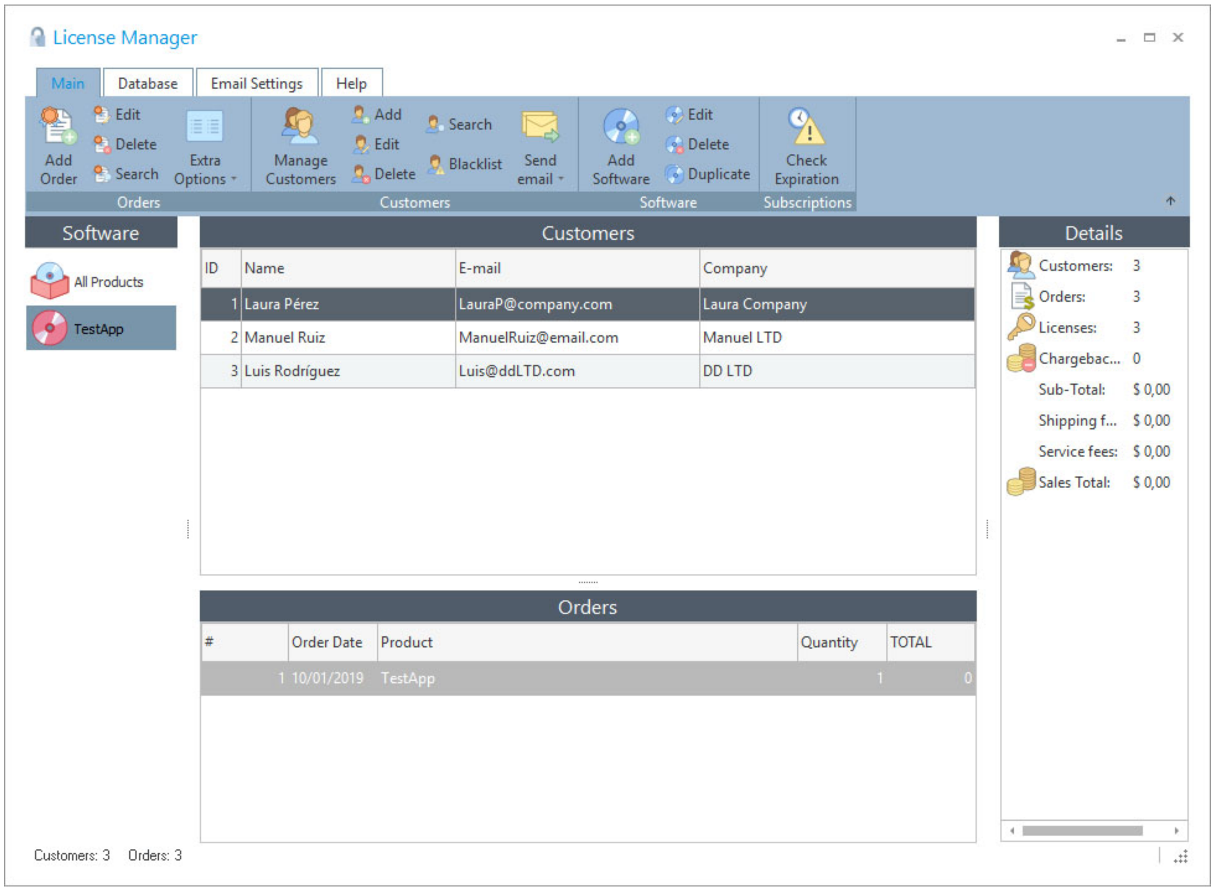
WinLicense includes a complete License Manager to help you managing your licenses, orders, software, customers, emails, subscriptions, etc.

The License Manager works with a MySQL database. You can work with the License Manager on a local PC (embedded database, single user) or with a server database (for multi user access in a remote machine).

2.1 User Interface

2.1.1 Main Panel

From the main application form, you can manage all your customers, software, orders, licenses, emails, etc.



ID	Name	E-mail	Company
1	Laura Pérez	LauraP@company.com	Laura Company
2	Manuel Ruiz	ManuelRuiz@email.com	Manuel LTD
3	Luis Rodríguez	Luis@ddLTD.com	DD LTD

#	Order Date	Product	Quantity	TOTAL
1	10/01/2019	TestApp	1	0

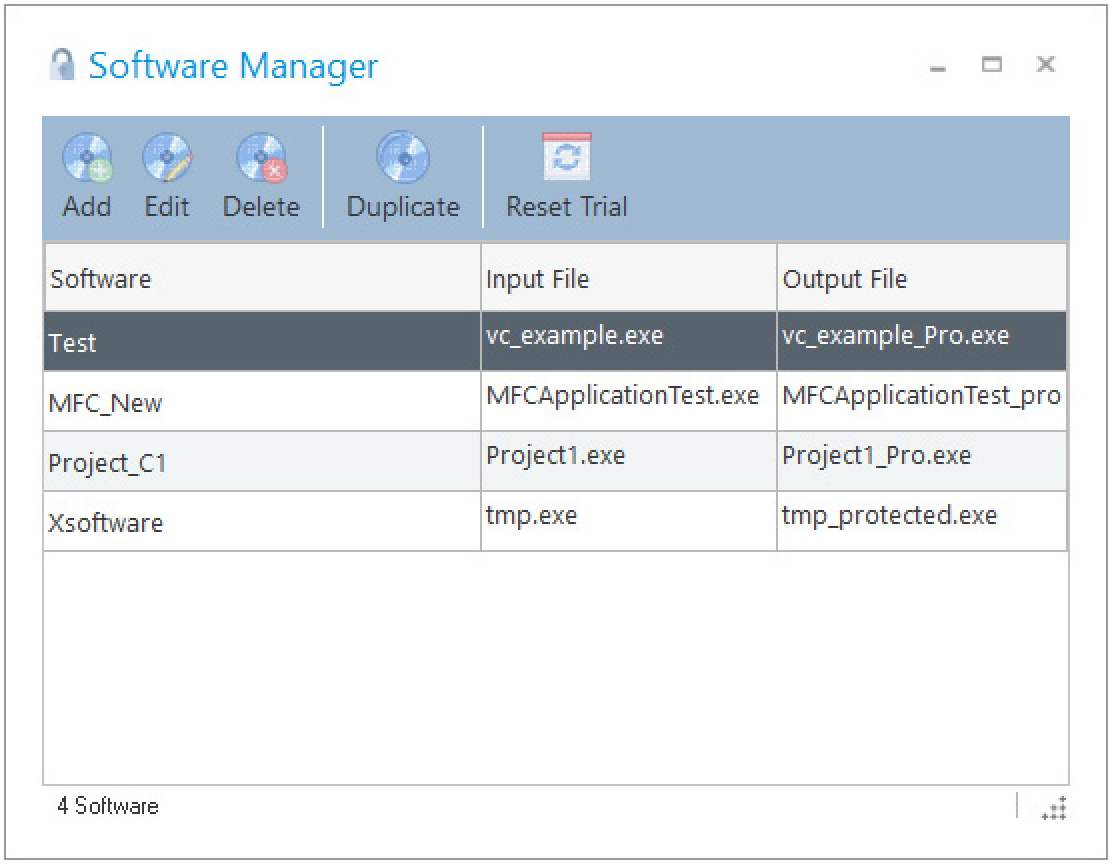
Customers:	3
Orders:	3
Licenses:	3
Chargebac...	0
Sub-Total:	$ 0,00
Shipping f...	$ 0,00
Service fees:	$ 0,00
Sales Total:	$ 0,00

Customers: 3 Orders: 3

2.1.2 Managing Software

Software Manager

From the software manager you can Add, Edit, Delete, Duplicate an Reset the trial of your software.



Software	Input File	Output File
Test	vc_example.exe	vc_example_Pro.exe
MFC_New	MFCApplicationTest.exe	MFCApplicationTest_pro
Project_C1	Project1.exe	Project1_Pro.exe
Xsoftware	tmp.exe	tmp_protected.exe

4 Software

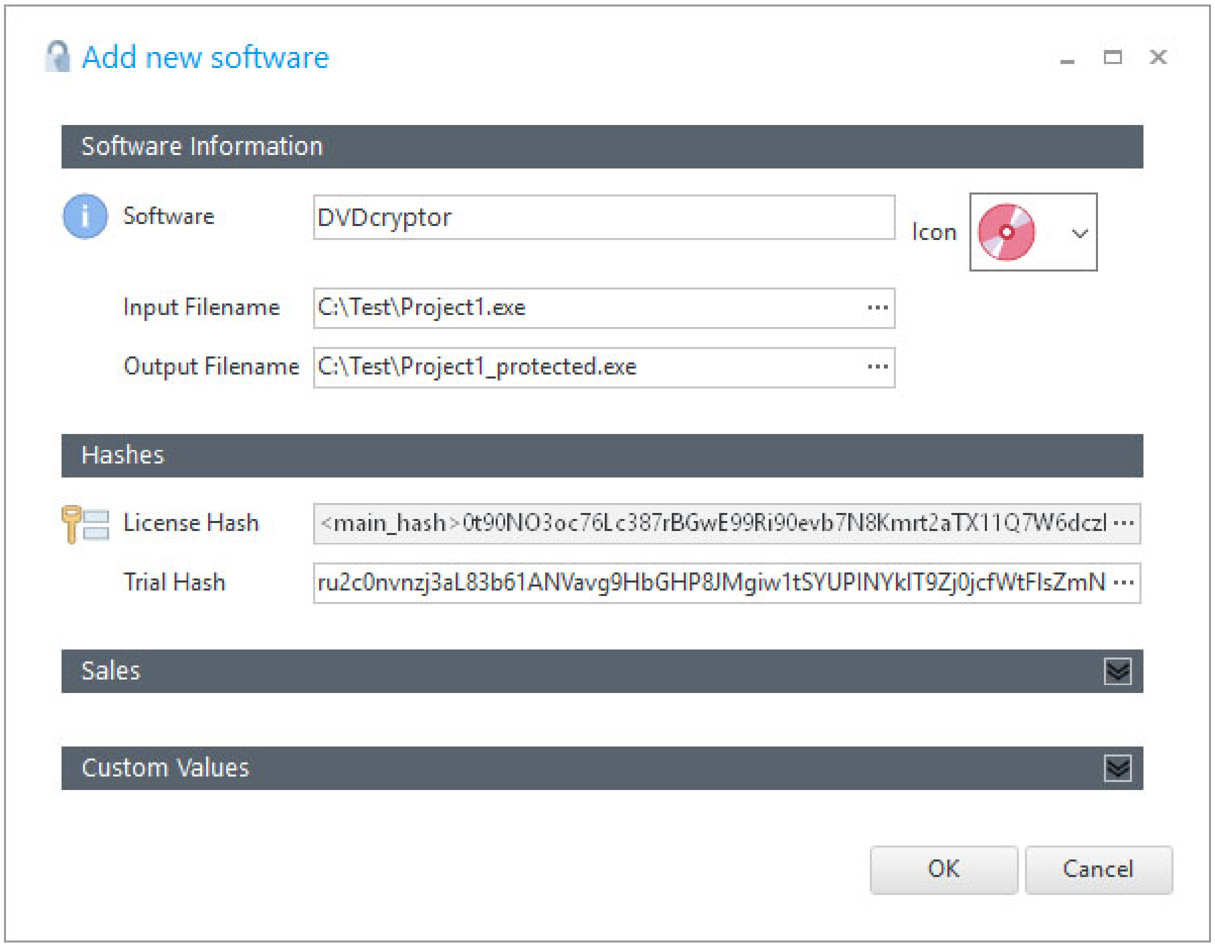
To Add/Edit a software, refer to the description below.

The **Duplicate** button makes a full copy of the selected software with a different name. When copying a software, the same hashes are copied, that means that the new software can be registered with the same licenses keys as the original software (meanwhile you don't change the hashes in the copied software)

The **Reset Trial** button resets the trial of the selected software. This will set a special flag in the system and when the protected software is launched the next time, the trial periods will be reset.

Adding a new software

When you add a new software, you just need to insert a name for the software and the rest of the fields are optional, just in case that you need them. You can also select an icon for your software, either from the drop down list or select a 48x48 bmp image that you have available.


Add new software

Software Information

Software: DVDcryptor

Icon

Input Filename: C:\Test\Project1.exe

Output Filename: C:\Test\Project1_protected.exe

Hashes

License Hash: <main_hash>0t90NO3oc76Lc387rBGwE99Ri90evb7N8Kmrt2aTX11Q7W6dczl

Trial Hash: ru2c0nvnzj3aL83b61ANVavg9HbGHP8JMgiw1tSYUPINYkIT9Zj0jcfWtFIsZmN

Sales

Custom Values

OK Cancel

Special Constants in Input/Output file names

Sometimes you might want to work with relative or not fixed paths in the Input and Output file names, so you can easily move your project files across different PCs or when working in a cooperative environment with several developers. You can use special defined constants as part of the file paths. This constants can be used in other places in the user interface that refers to a file path (like the XBundler files, Splash file, etc.). The current available constants are the following:

- **%INPUT_FILE_FOLDER%**: Specifies the folder of the input file (the file to protect)

- **%OUTPUT_FILE_FOLDER%**: Specifies the folder of the output file (the protected file)

- **%WINLICENSE_FOLDER%**: Specifies the folder where WinLicense.exe (WinLicense64.exe) is located

- **%CURRENT_FOLDER%**: Specifies the current folder from where WinLicense.exe has been launched

- ***%environment_variable%***: Specifies the folder defined in any environment variable. For example, if you want to get the path from the "temp" environment variable, you can write something like: *%temp%\MyApplication.exe*

An example of an input file to protect can be**:** *%WINLICENSE_FOLDER%\MyProjects\MyApplication.exe*

Hashes

The WinLicense hashes is the base to make licenses unique for a specific application (**License Hash**) and to make the trial to be independent for each protected application (**Trial Hash**)

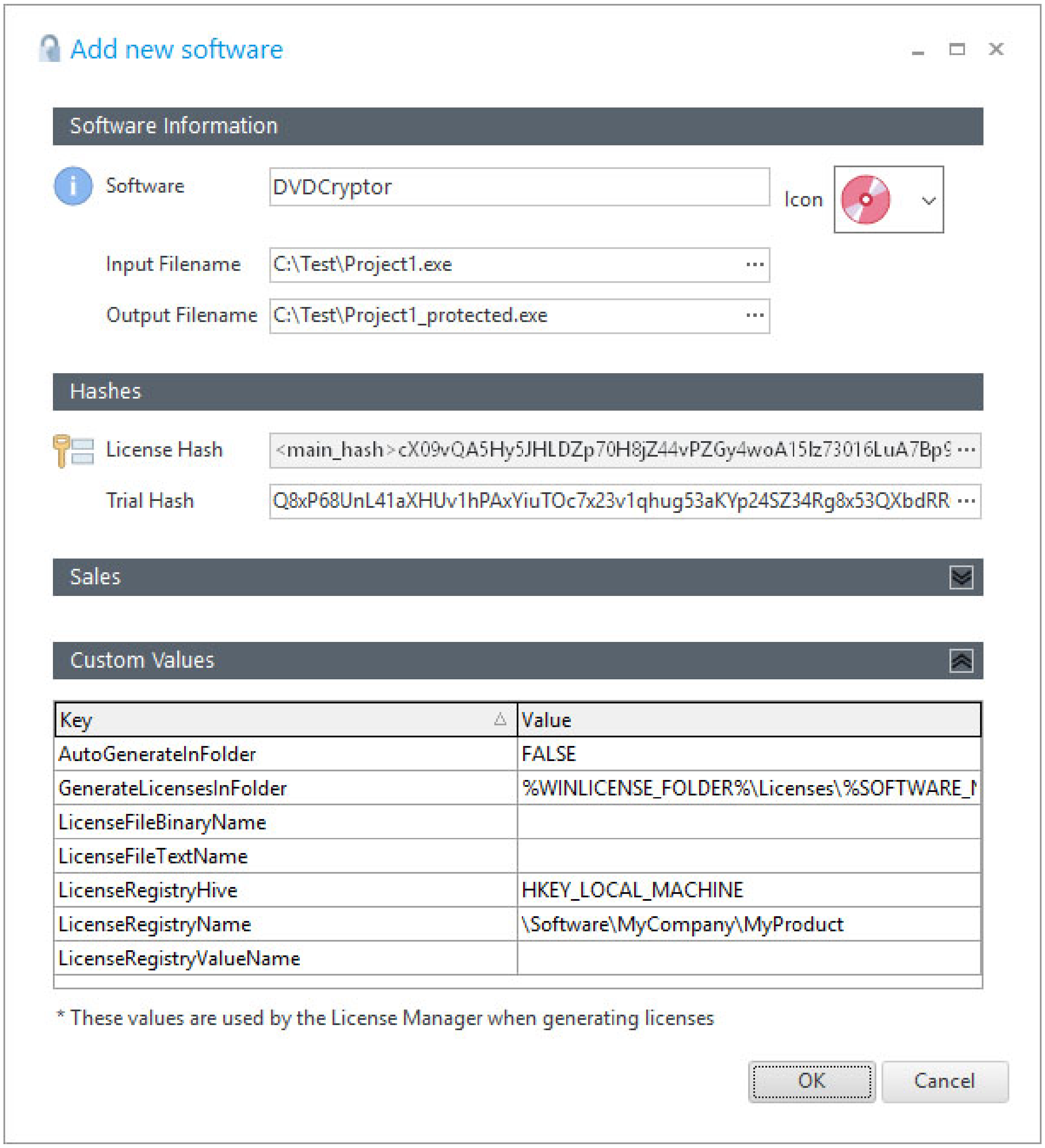
- **License Hash:** This field specifies the hash key that will be used to generate specific licenses for the current software. If two applications are protected with **different** License Hash, generated license keys for the first application will be invalid for the second application. In the same manner, if several applications are protected with the **same** License Hash, generated licensed for a single application will be valid for the rest of applications with that same License Hash. For example, suppose that you have your application (MyApp.exe) with version 1.0 which accepts file license keys (regkey.dat). Now you are going to deliver version 2.0 and you don't want that the previous regkey.dat for version 1.0 works with the new version 2.0, you just need to regenerate a new License Hash for your version 2.0 and the previous regkey.dat won't be valid for version 2.0

- **Trial Hash**: This field specifies the hash key that will be used to store the trial status for the protected software. Suppose that you have two applications protected with the same Trial Hash and both of them expire by number of days, in this case the number of days left will match in both protected applications, when there no more days left, both applications are expired. On the other hand, if you want that each application has its own trial expiration status, you just need to generate a different Trial Hash for each protected application. Another example: if you protect an application "A" (version 1.0) with a trial hash and at a later time you protect a new version of "A" (version 2.0), the trial period will start as fresh (like reset) for the new version. That is, if version 1.0 was expired on a specific client's computer, the new version 2.0 will start running on that same computer like if the previous version was not installed before.

Custom Values

The Custom Values panel offers lots of flexibility to add special options for a specific software. The Custom Values are normally relate with the license generation, to specify the license name, folder location, etc. When you generate a new Software, you can see that there are some default custom values that you can fill to match your needs when generating a new license.

To create a new entry, just right-click on that panel and a floating menu will appear with the new Key name to insert.



Add new software

Software Information

Software: DVDCryptor

Icon

Input Filename: C:\Test\Project1.exe

Output Filename: C:\Test\Project1_protected.exe

Hashes

License Hash: <main_hash>cX09vQA5Hy5JHLDZp70H8jZ44vPZGy4woA15Iz73016LuA7Bp9

Trial Hash: Q8xP68UnL41aXHUv1hPAxYiuTOc7x23v1qhug53aKYp24SZ34Rg8x53QXbdRR

Sales

Custom Values

Key	Value
AutoGenerateInFolder	FALSE
GenerateLicensesInFolder	%WINLICENSE_FOLDER%\Licenses\%SOFTWARE_I
LicenseFileBinaryName	
LicenseFileTextName	
LicenseRegistryHive	HKEY_LOCAL_MACHINE
LicenseRegistryName	\Software\MyCompany\MyProduct
LicenseRegistryValueName	

* These values are used by the License Manager when generating licenses

Here we present a simple example:

1) You can create a key-value "LicenseFileBinaryName" and write as value the name of the expected license file name that will register your application.

2) From the emails template manager, you can use the constant *$LicenseFileBinaryName* that will be replaced with the name of the file that you set. For example, you can create an email template as:

Dear ($User_Name),

Please, find attached your license ($LicenseFileBinaryName).

Best Regards,

The Sales Team

The following table describes the default Custom Values for a Software:

Key	Description
AutoGenerateInFolder	If set to YES/TRUE each time that you generate a license from the Orders panel, it will be generated into a file
GenerateLicensesInFolder	From the Orders panel, you can right-click on the displayed license and export the license to a file. This variable contains the default folder for the Windows Shell SaveDialog.
LicenseFileBinaryName	This is the expected license file name in case that you are registering your application via a Single File. This name should match the one that you have in the Registration panel from your WinLicense Project. Please, refer below for special folder constants that can be used.
LicenseFileTextName	This is the expected license file name in case that you are generating licenses in text format. Please, refer below for special folder constants that can be used.
LicenseRegistryHive	This is the Registry key hive that is expected if you are generating registry key based licenses. This value should match with the Registry Key hive that you have in the Registration panel in your WinLicense Project.

LicenseRegistryName	This is the Registry key name that is expected if you are generating registry key based licenses. This value should match with the Registry Key hive that you have in the Registration panel in your WinLicense Project.
LicenseRegistryValueName	This is the Registry key value name that is expected if you are generating registry key based licenses. This value should match with the Registry Key hive that you have in the Registration panel in your WinLicense Project.
OpenFolderWhenGeneratingLicense	If set to YES/TRUE it will open the folder in Explorer when exporting a file on disk.
DefaultRegistrationType	Default type of license to generate. Possible values are **FILE** (file key), **REGISTRY** (registry key), **TEXT** (text key), **STATIC** (static smartkey), **DYNAMIC** (dynamic smartkey)
AutomaticSelectLastCustomer	In the License Manager, it will select the last customer that did an order when selecting the specific software
LicenseSmartKeyAppendHeader	If set to YES/TRUE it will append a header and a tail to the generated SmartKey. This is to prevent invisible chars addition when the SmartKey is sent via email and copied by the final customer

Special Constants to be used as part of File/Folder names

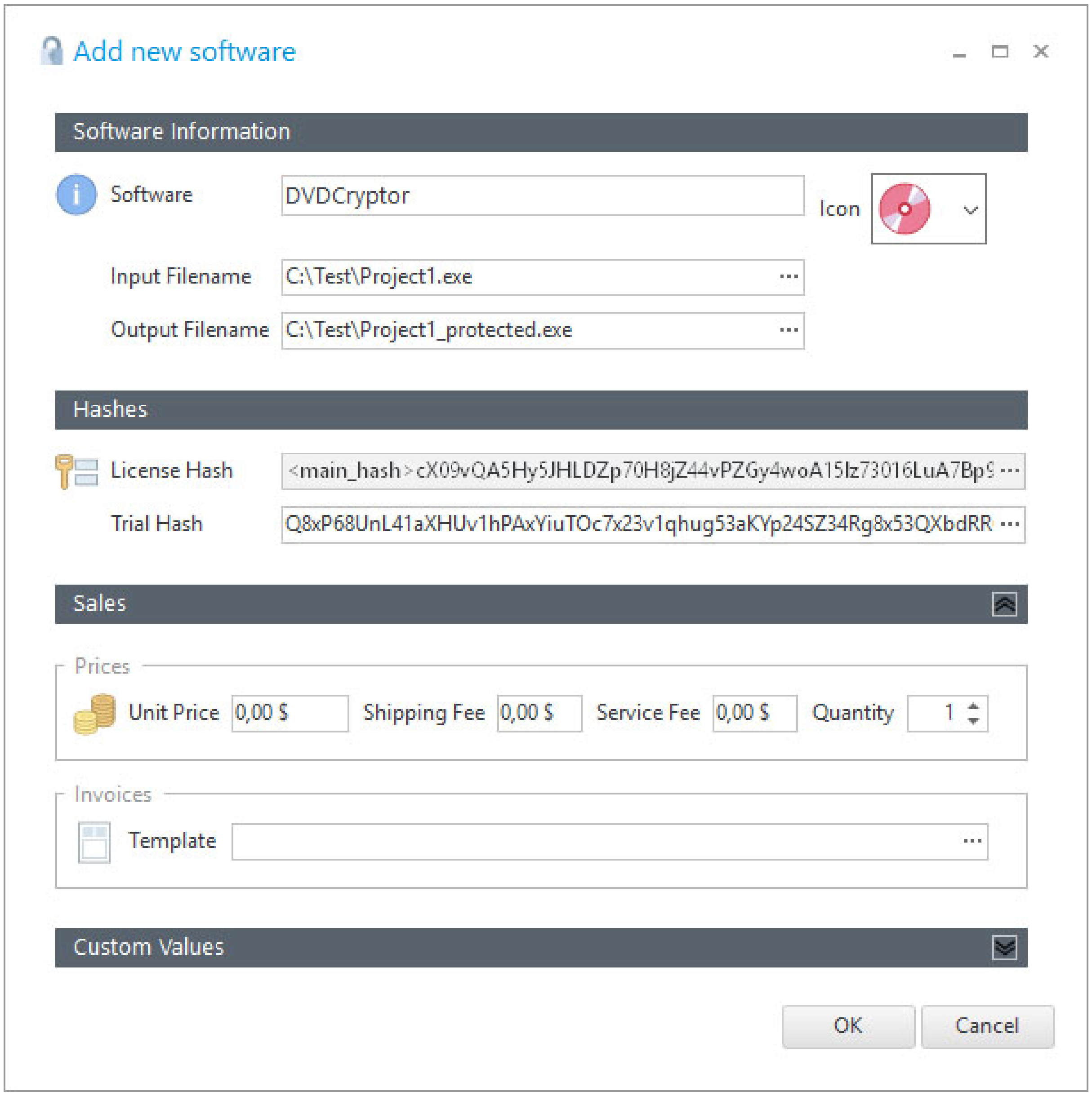
You can use some of the predefined special constants as part of specific Custom Values related with file and folder names. This apply for example to the Custom Values: *GenerateLicenseInFolder*, *LicenseFileBinaryName* and *LicenseFileTextName*.

- **%SOFTWARE_NAME%** or **%SOFT_NAME%**: Specifies the current Software name
- **%CUSTOMER_FIRST_NAME%**: Specifies the customer's first name
- **%CUSTOMER_LAST_NAME%**: Specifies the customer's last name
- **%CUSTOMER_COMPANY%**: Specifies the customer's company
- **%CURRENT_FOLDER%**: Specifies the current folder from where WinLicense.exe was launched from

- **%INPUT_FILE_FOLDER%**: Specifies the folder for the input file to protect
- **%OUTPUT_FILE_FOLDER%**: Specifies the folder for the output (protected) file
- **%WINLICENSE_FOLDER%**: Specifies the folder where WinLicense.exe is located
- **%LICENSE_USER_NAME%**: Specifies the user name that appears in the generated license
- **%LICENSE_USER_COMPANY%**: Specifies the company name that appears in the generated license
- **%LICENSE_HARDWARE_ID%**: Specifies the company name that appears in the generated license
- **%LICENSE_DAYS_EXPIRATION%**: Specifies the number of days expiration in the generated license
- **%LICENSE_EXECUTIONS_EXPIRATION%**: Specifies the number of executions expiration in the generated license
- **%LICENSE_DATE_EXPIRATION%**: Specifies the expiration date in the generated license
- **%LICENSE_RUNTIME_EXECUTION%**: Specifies the license runtime execution
- **%LICENSE_GLOBAL_TIME%**: Specifies the license global time exeuction
- **%LICENSE_NETWORK_INSTANCES%**: Specifies the number of network instances in the generated license
- **%LICENSE_LOCKED_COUNTRY%**: Specifies the country where the license was locked to

Sales

If you want to keep track of your sales, you can specific the price and deliver costs for your software. You can also define an invoice template to be used when you generate invoices for your orders. The invoice template should be a DOC document where you can use predefined constants that will be replaced for each specific order.


Add new software

Software Information

Software: DVDCryptor
Icon

Input Filename: C:\Test\Project1.exe

Output Filename: C:\Test\Project1_protected.exe

Hashes

License Hash: <main_hash>cX09vQA5Hy5JHLDZp70H8jZ44vPZGy4woA15lz73016LuA7Bp

Trial Hash: Q8xP68UnL41aXHUv1hPAxYiuTOc7x23v1qhug53aKYp24SZ34Rg8x53QXbdRR

Sales

Prices

Unit Price: 0,00 $ Shipping Fee: 0,00 $ Service Fee: 0,00 $ Quantity: 1

Invoices

Template:

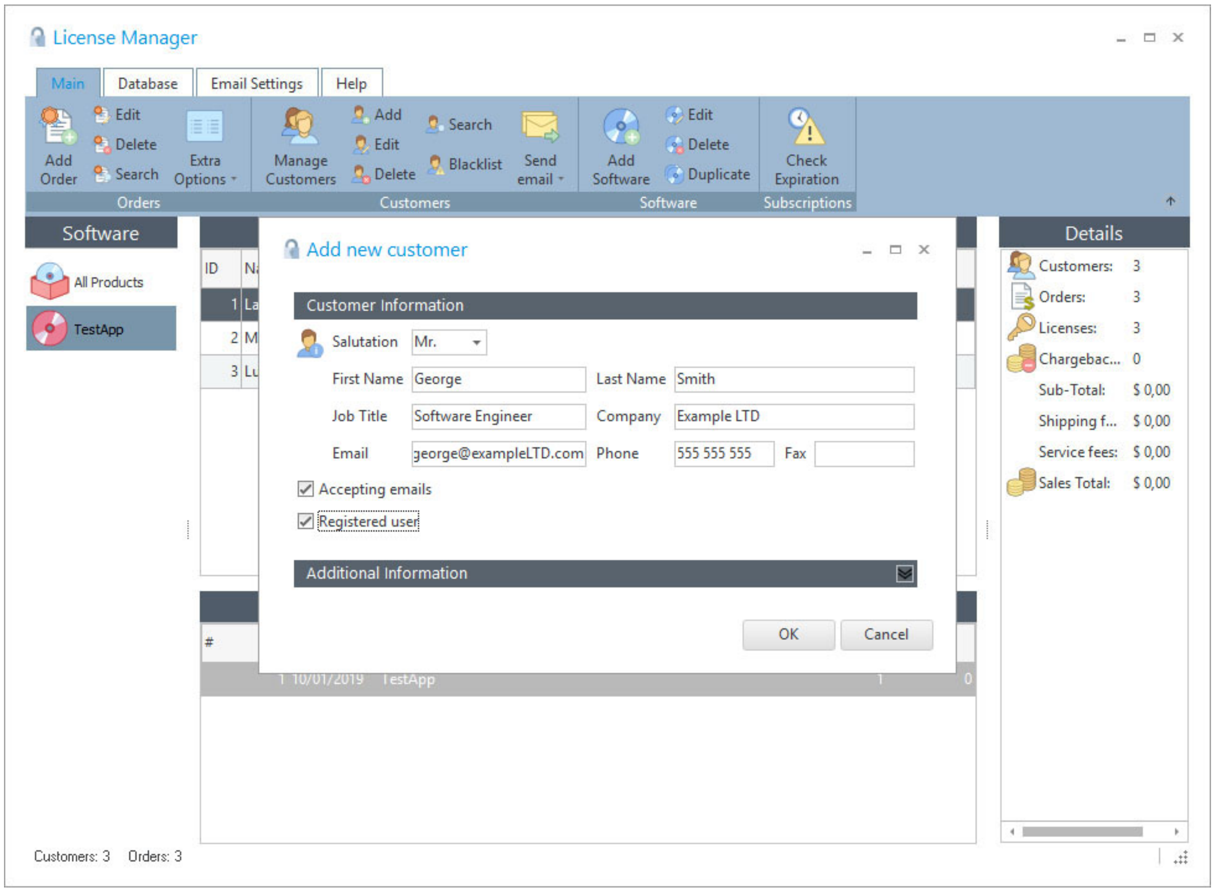
Custom Values

2.1.3 Managing Customers

From the main application form, you can Add, Edit, Find and Remove Customers.

Add a new Customer

When you add a new customer the minimun information required is the "Name" and "Last Name" fields. If you plan to send emails to that customer at a later time you should specify a valid email address for that customer.



Add new customer

Customer Information

Salutation	Mr.		
First Name	George	Last Name	Smith
Job Title	Software Engineer	Company	Example LTD
Email	george@exampleLTD.com	Phone	555 555 555
Fax			

☑ Accepting emails

☑ Registered user

Additional Information

OK Cancel

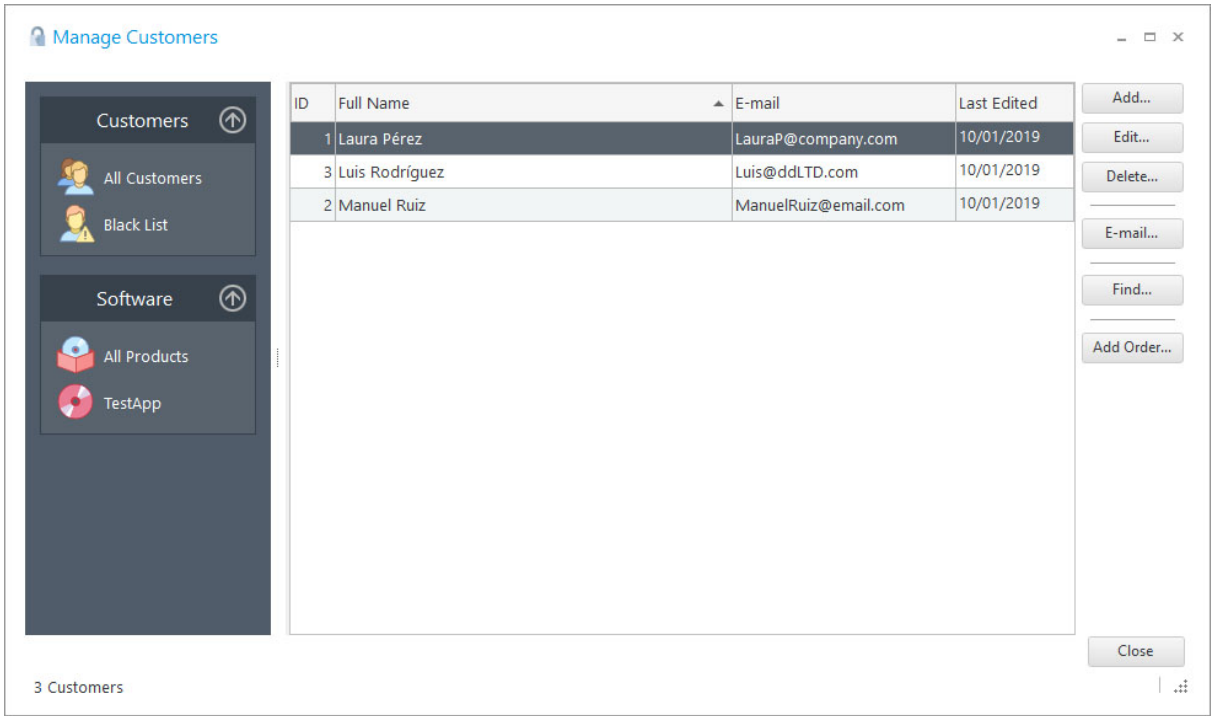
There are two special fields for each Customer:

1) The "Accepting e-mails" field should be checked if you plan to send emails to that user at a later time. For example, you select one software from the side bar and your customers are displayed, if you send an email to "All", those customers with the "Accepting e-mails" field *unchecked* will not receive the email. Basically, that field should be set if you don't plan to keep communication with that client either temporally or permanently. For example if from the "Check Orders Expiration" panel you see that an order has expired for a specific client, you send him a last email notificating him about the expiration and after that you uncheck the field "Accepting e-mails".

2) The "Registered user" field can be used to know that the current user is a valid user which is currently active. When you send an email to All Customers using a filter (for a single or all your software), you can see that you can specify if the email will be sent to customers with the "Registered User" field set, unset or both. There is not a standard way about when you should set/unset this field, you should use it in the way that it adapts better to your specific scenario.

Manage Customers

From this form you can Add, Edit, Find, Remove and Send emails to your customers. You can select a specific software and view all customers that purchased that software and send emails to a specific or all customers that appears on the list or send a email to a filtered group of customers.

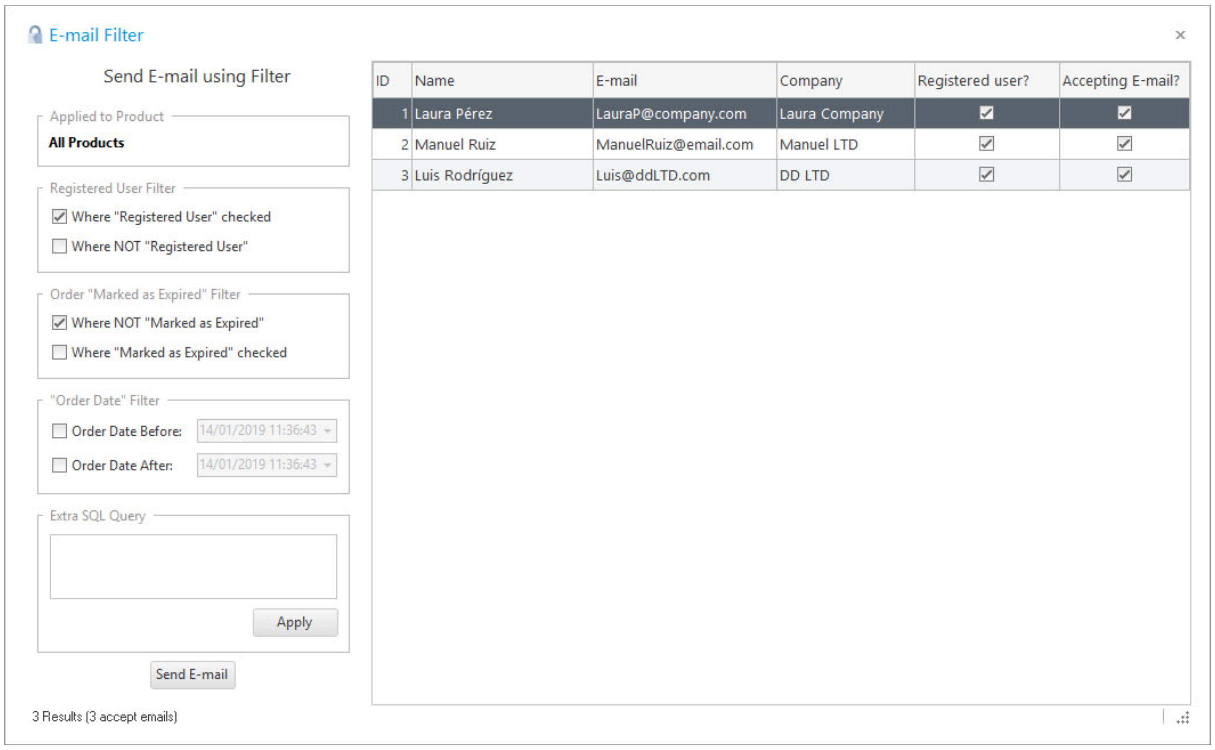


ID	Full Name	E-mail	Last Edited
1	Laura Pérez	LauraP@company.com	10/01/2019
3	Luis Rodríguez	Luis@ddLTD.com	10/01/2019
2	Manuel Ruiz	ManuelRuiz@email.com	10/01/2019

This panel is helpful when you want to communicate with your clients via email. You can select the "All Customers" icon or specify a software icon and all your customers will be displayed or customer for a specific software. After that, you can send emails to all or applying a filter.

Send E-mails using Filter

After selecting "All Customers" or specifying a software in the "Customer Manager" panel, all collected clients can be filtered, so emails are sent to a specific group of clients.



Applied to Product: All Products

[✓] Where "Registered User" checked
[ ] Where NOT "Registered User"

[✓] Where NOT "Marked as Expired"
[ ] Where "Marked as Expired" checked

[ ] Order Date Before: 14/01/2019 11:36:43
[ ] Order Date After: 14/01/2019 11:36:43

ID	Name	E-mail	Company	Registered user?	Accepting E-mail?
1	Laura Pérez	LauraP@company.com	Laura Company	✓	✓
2	Manuel Ruiz	ManuelRuiz@email.com	Manuel LTD	✓	✓
3	Luis Rodríguez	Luis@ddLTD.com	DD LTD	✓	✓

3 Results (3 accept emails)

The following filters can be used:

- **Where "Registered User" checked**: Customers with the "Registered User" field (in the Customers table) are displayed.

- **Where NOT "Registered User"**: Customers with the "Registered User" field unchecked (in the Customers table) are displayed.

- **Where NOT "Marked as Expired"**: Customers where the "Marked as Expired" field (in the current software order) is not set, are displayed.

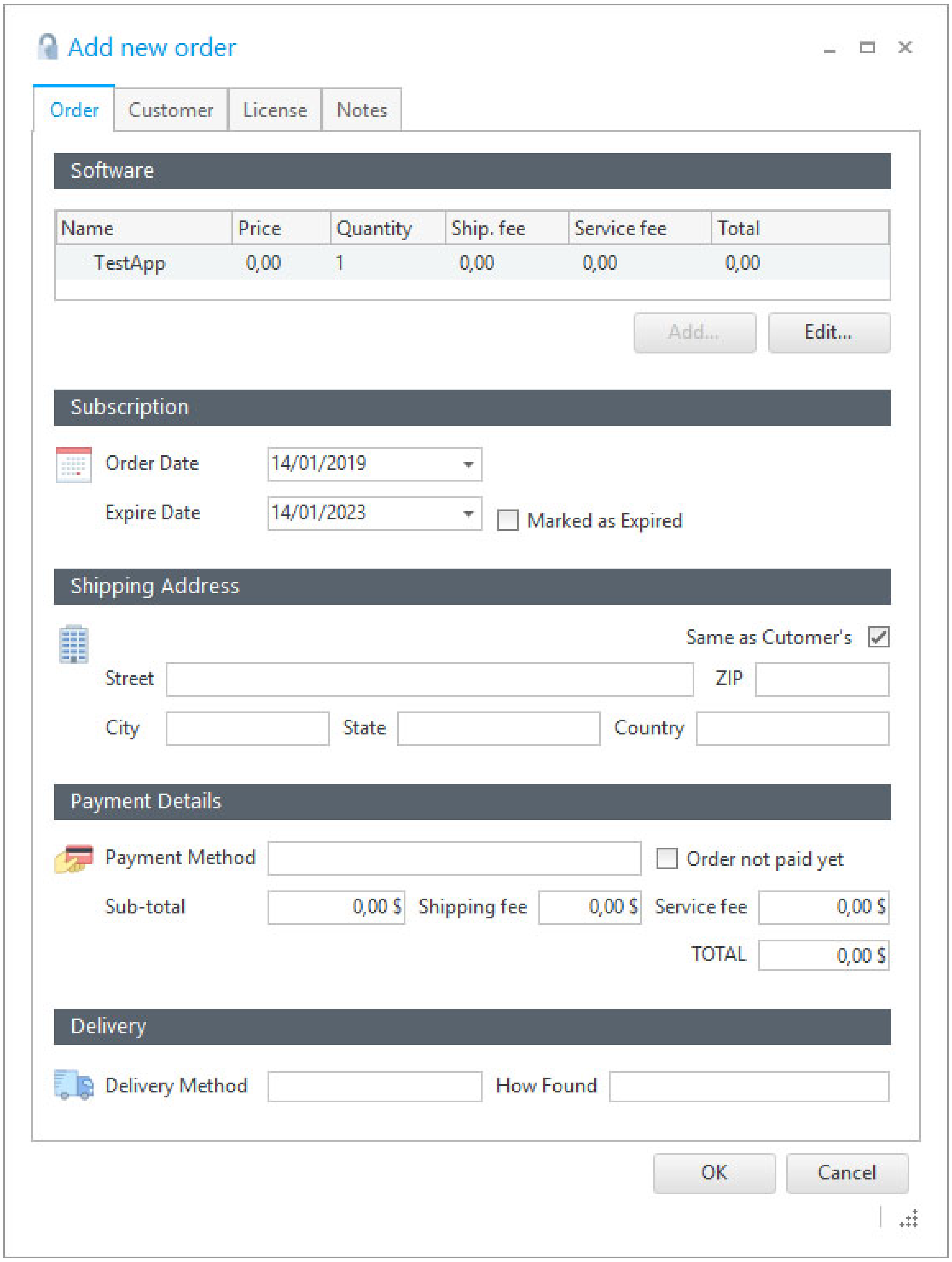
- **Where "Marked as Expired" checked**: Customers where the "Marked as Expired" field (in the current software order) is set, are displayed.

2.1.4 Managing Orders

From the main application form, you can Add, Edit, Find and Remove orders.

Adding a new order

To add a new order, basically you just need to **select a software** (the one that is being sold) **and a customer** (the one that has purchased the software), nothing else is required. The rest of the fields are optional and can be used if required. Notice that in order to create a new order, you first need to enter the customer into the database.



Name	Price	Quantity	Ship. fee	Service fee	Total
TestApp	0,00	1	0,00	0,00	0,00

Order Date 14/01/2019

Expire Date 14/01/2023 ☐ Marked as Expired

Same as Cutomer's ✓

Street ZIP

City State Country

Payment Method ☐ Order not paid yet

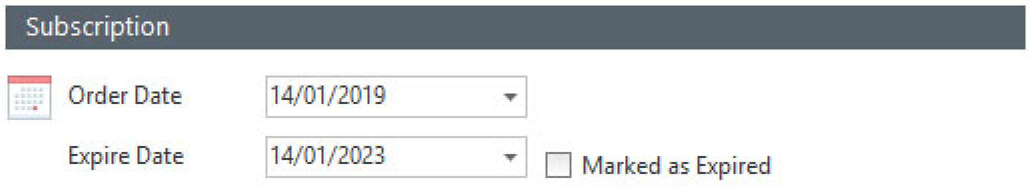
Sub-total 0,00 $ Shipping fee 0,00 $ Service fee 0,00 $

TOTAL 0,00 $

Delivery Method How Found

Expiration information in Order

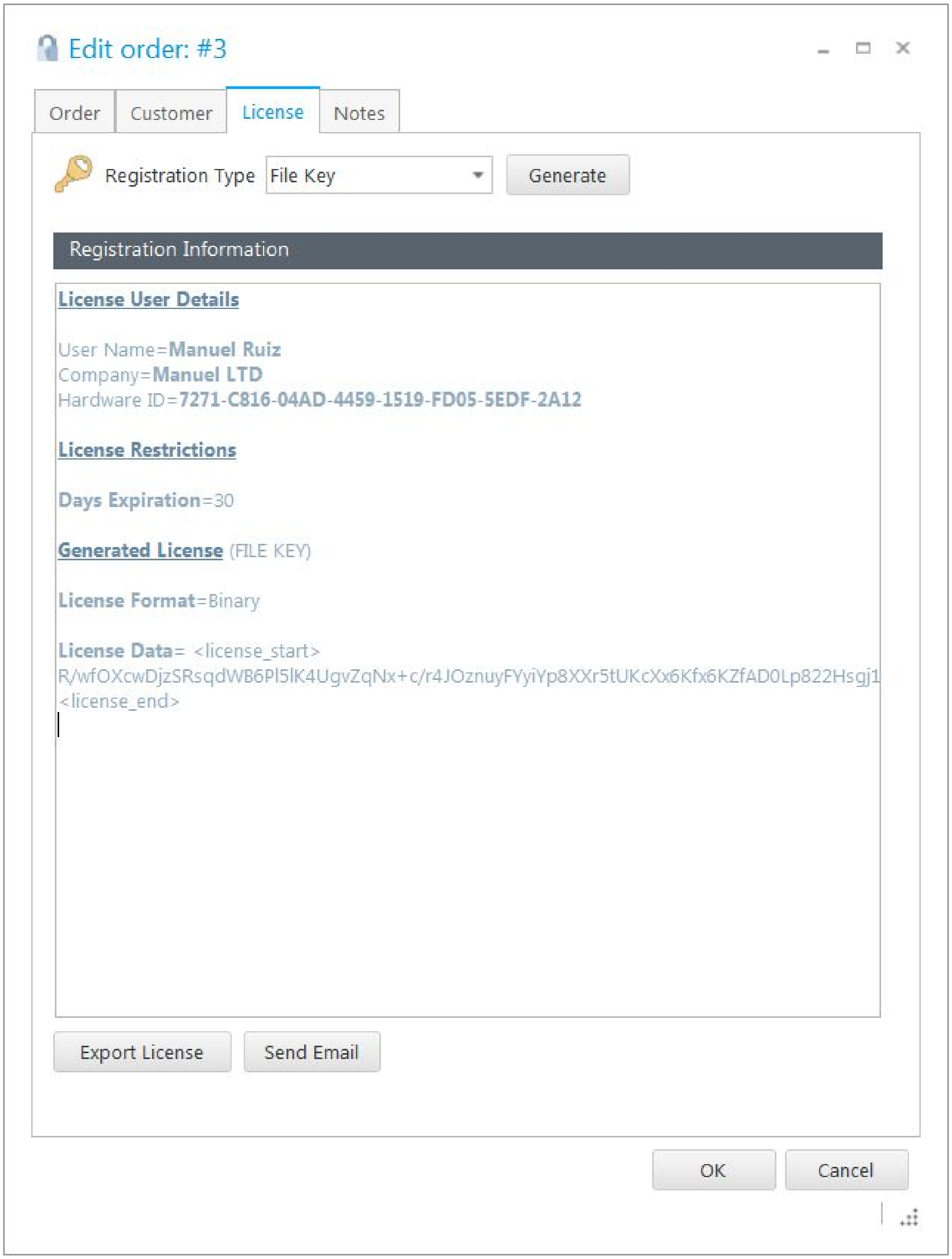
If you are selling licenses that will expire at a specific date (or subscription plan), then you can make use of the "Expiration Date" field when you create a new license. From the "Check Orders Expiration" panel, you can check which licenses are expired or will expire within the next days.



The "Marked as expired" field is a special field when working with Expiration Dates. Basically that field can be used when an order/license/subscription has expired and you have already taken action of that. For example, from the "Check Orders Expiration" panel you see that a specific order has expired and you send an email to that client mentioning about the expiration. After that, you can check the "Mark as expired" field into that order to specify that the customer has being notified about the expiration and you fully mark the order as expired. When sending emails to a group of customers, you can specify if the email will be sent to customers with the "Marked as expired" field set, unset or both. There is not a standard way about when you should set/unset that field, you should use it in the way that it adapts better to your specific scenario.

License Generation

In the "License" tab you can see that you can generate licenses for WinLicense applications or for other applications. When you generate licenses for a WinLicense application, you just select the type of license expected by your application (File, Registry, SmartKey 93...) and click on the Generate button 391. After that a new form appears with all the available WinLicense restrictions that you can insert in the license to generate.



Registration Type File Key

License User Details

User Name=**Manuel Ruiz**
Company=**Manuel LTD**
Hardware ID=**7271-C816-04AD-4459-1519-FD05-5EDF-2A12**

License Restrictions

Days Expiration=30

Generated License (FILE KEY)

License Format=Binary

License Data= <license_start>
R/wfOXcwDjzSRsqdWB6Pl5lK4UgvZqNx+c/r4JOznuyFYyiYp8XXr5tUKcXx6Kfx6KZfAD0Lp822Hsgj1
<license_end>

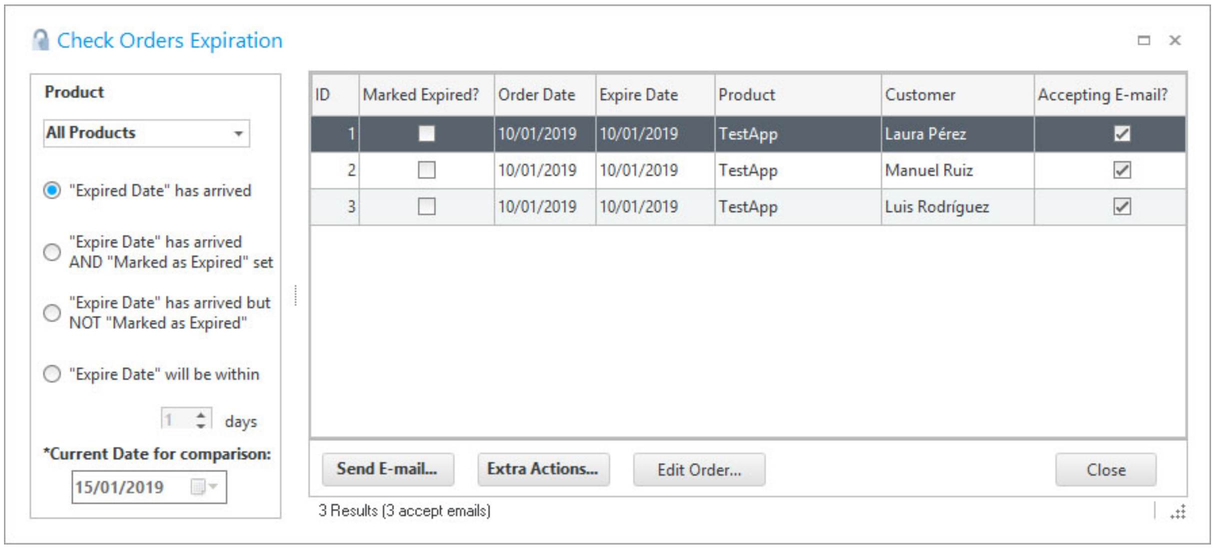
After the license is generated, the **Registration Content** panel will contain the generated license info under the **License Data** key pair. This license is already stored inside the WinLicense database and you can, for example, email 379 that license to your customer.

The "Export License" button allows you to export the license into a file, so you can grab it from the database and test it with your protected application.

The "Send Email" button allows you to inmediatelly send the generated license to your customer, using any of your predefined email templates 379.

2.1.5 Check Expiration

The Check Expiration form can be accessed from the main application form. This form will present a list of orders that are already expired or are about to expire within a number of days. After your apply the wanted filter, you can send an email to all displayed customers and/or do some extra options (from the "Extra Actions" button) for the displayed orders.


Check Orders Expiration
Product
All Products
"Expired Date" has arrived
"Expire Date" has arrived AND "Marked as Expired" set
"Expire Date" has arrived but NOT "Marked as Expired"
"Expire Date" will be within
1
days
*Current Date for comparison:
15/01/2019
Send E-mail...
Extra Actions...
Edit Order...
Close
3 Results (3 accept emails)

ID	Marked Expired?	Order Date	Expire Date	Product	Customer	Accepting E-mail?
1	☐	10/01/2019	10/01/2019	TestApp	Laura Pérez	✓
2	☐	10/01/2019	10/01/2019	TestApp	Manuel Ruiz	✓
3	☐	10/01/2019	10/01/2019	TestApp	Luis Rodríguez	✓

The date comparison is performed by taken the system current time and comparing it with the "Expire Date" field that you specified when creating/editing each order.

This form is presented with a side panel that works as a filter for the presented results.

- **Software**: You can specify if you are checking expiration for all your software or a specific software.

- **"Expire Date" has arrived**: Orders for the selected software, where the "Expire Date" is minor than the current date (taken from system time), are displayed

- **"Expire Date" has arrived AND "Marked as Expired" set**: Orders for the selected software, where the "Expire Date" is minor than the current date and you have checked the "Marked as Expired" field, are displayed.

- **"Expire Date" has arrived but NOT "Marked as Expired"**: Orders for the selected software, where the "Expire Date" is minor than the current date and you have not checked the "Marked as Expired" field, are displayed.

- **"Expire Date" will be within xx days**: Orders for the selected software, where the "Expire Date" is minor than the current date plus the selected number of days, are displayed.

Extra Actions

After applying a filter, apart from sending an email, you can perform specific options for the displayed records (by clicking on the "Extra Actions..." button)

- **Order --> Set "Marked as Expired"**: You can set the "Marked as Expired" field (in the Orders table) for the selected order or for all displayed orders.

- **Order --> Extend "Expire Date"**: You can extend the "Expire Date" field (in the Orders table) for the selected order or for all displayed orders.

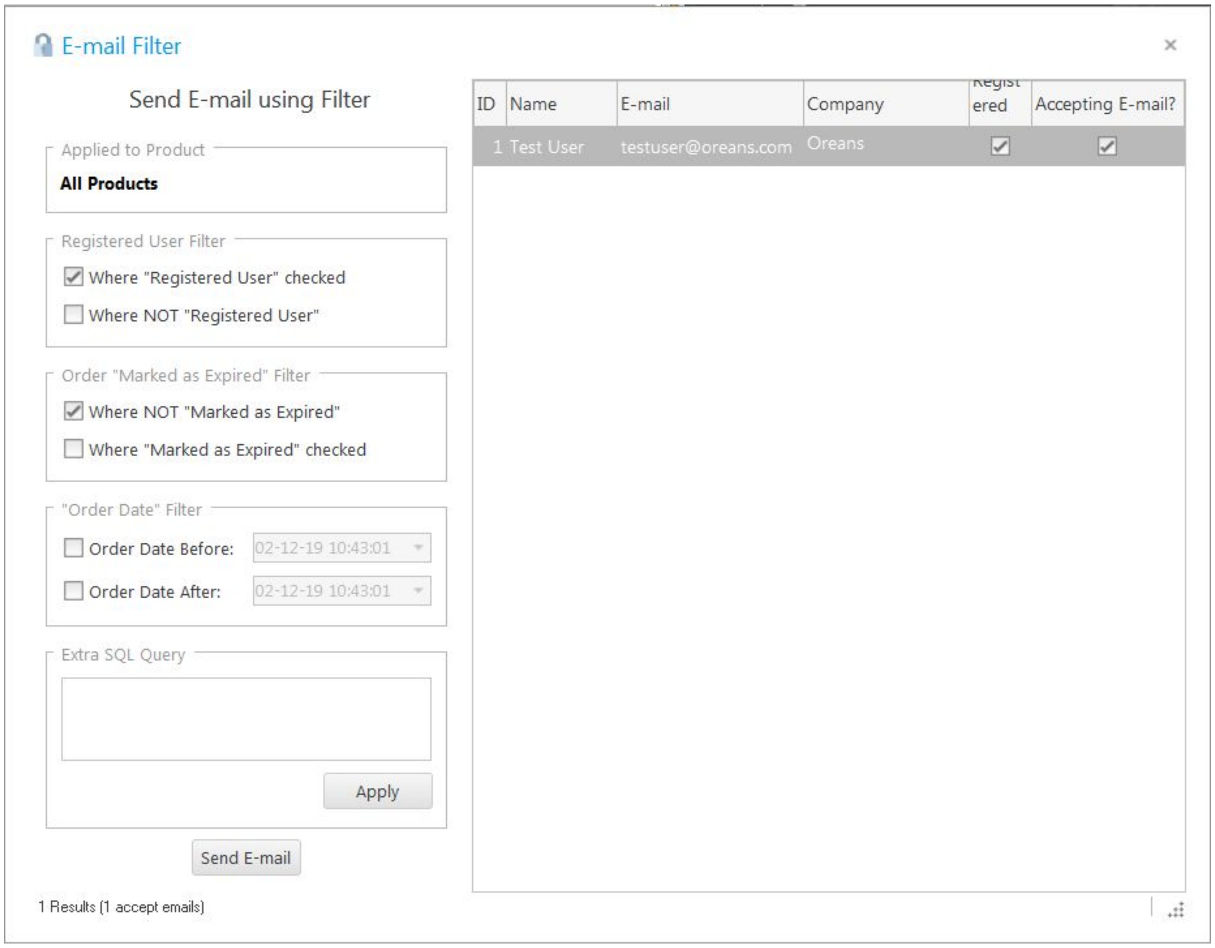
- **Customer --> Unset "Registered User"**: You can uncheck the "Registered User" field (in the Customer table) for a customer in the selected order, or for all customers for each displayed order.

- **Customer --> Unset "Accepting e-mails"**: You can uncheck the "Accepting e-mails" field (in the Customer table) for a customer in the selected order, or for all customers for each displayed order.

2.1.6 E-mails

Send emails using filter

From the main License Manager panel, you cand send an email to specific customers (from a filter). From the top menu "**Main --> Send email --> to all using filter**".



E-mail Filter

Send E-mail using Filter

Applied to Product: All Products

Registered User Filter
- [✓] Where "Registered User" checked
- [ ] Where NOT "Registered User"

Order "Marked as Expired" Filter
- [✓] Where NOT "Marked as Expired"
- [ ] Where "Marked as Expired" checked

"Order Date" Filter
- [ ] Order Date Before: 02-12-19 10:43:01
- [ ] Order Date After: 02-12-19 10:43:01

Extra SQL Query

ID	Name	E-mail	Company	Registered	Accepting E-mail?
1	Test User	testuser@oreans.com	Oreans	✓	✓

1 Results (1 accept emails)

You can select specific filters to determine the clients that you want to send the email to:

- **Where "Registered User" checked**: When adding/editing a customer, there is a "Registered user" checkbox. This option filters customers that have that checkbox checked.

- **Where NOT "Registered User"**: When adding/editing a customer, there is a "Registered user" checkbox. This option filters customers that have that checkbox unchecked.

- **Where NOT "Marked as Expired"**:When adding/editing an order, there is a "Marked as Expired" checkbox. This option filters orders that have that checkbox unchecked.

- **Where "Marked as Expired" unchecked**: When adding/editing an order, there is a "Marked as Expired" checkbox. This option filters orders that have that checkbox checked.

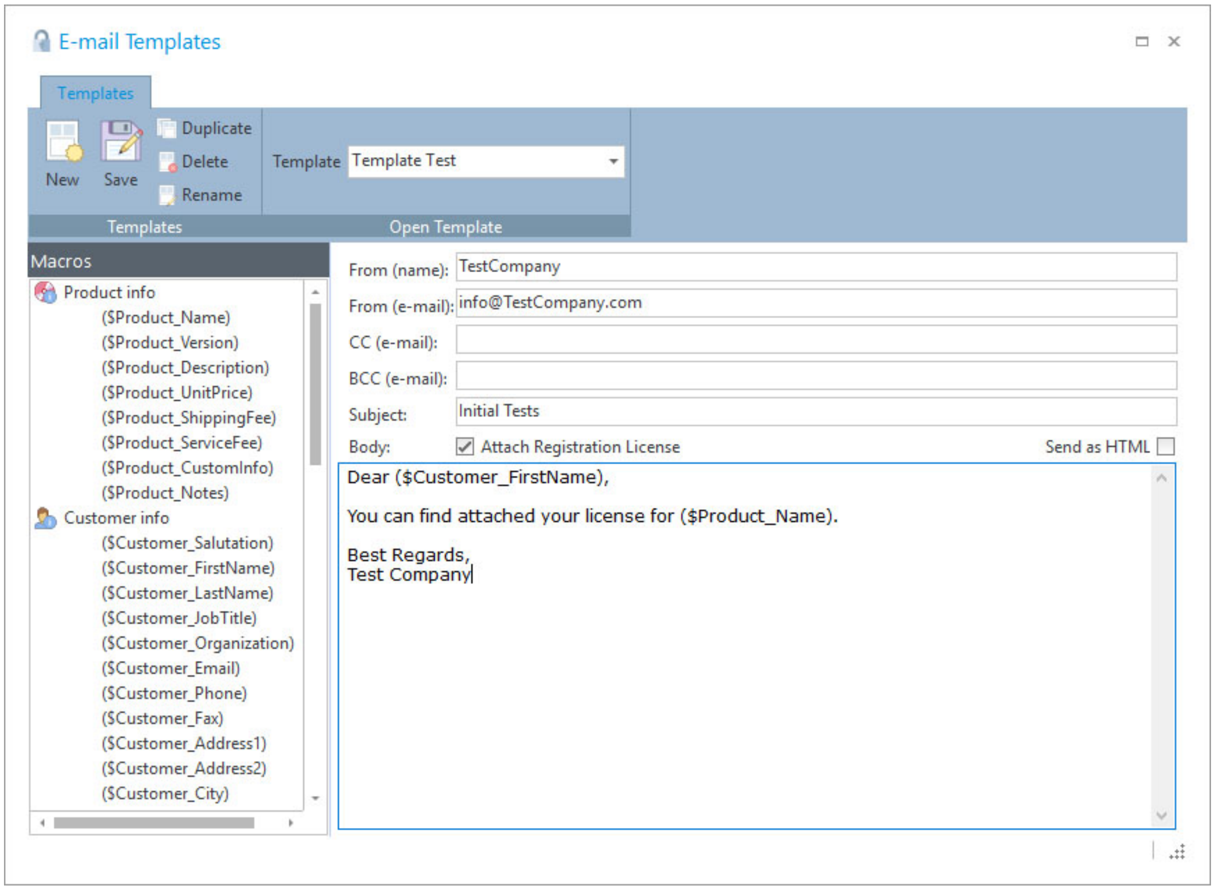
- **Order Date Before **date****: When adding/editing an order, there is an "Order Date" checkbox. This options filters orders that have an "order date" before the specified date.

- **Order Date After **date****: When adding/editing an order, there is an "Order Date" checkbox. This options filters orders that have an "order date" after the specified date.

- **Extra SQL Query**: The selected items above will generate a SQL query to retrieve the desired entries, with this option you can extend the SQL query. For example, if you also want to filter for orders that have the string "premium" you can add the following entry: ***`AND Order_RegistrationContent LIKE '%premium%'`***
  To know more about the specific table and column names used by WinLicense, you can go to the top menu "Database --> Manage".

Template Manager

The E-mails Manager can be accessed from the top menu "**E-Mails Settings-->Template Manager**". From this form, you can create emails template to be selected when you are going to send specific emails to your clients.



Template: Template Test

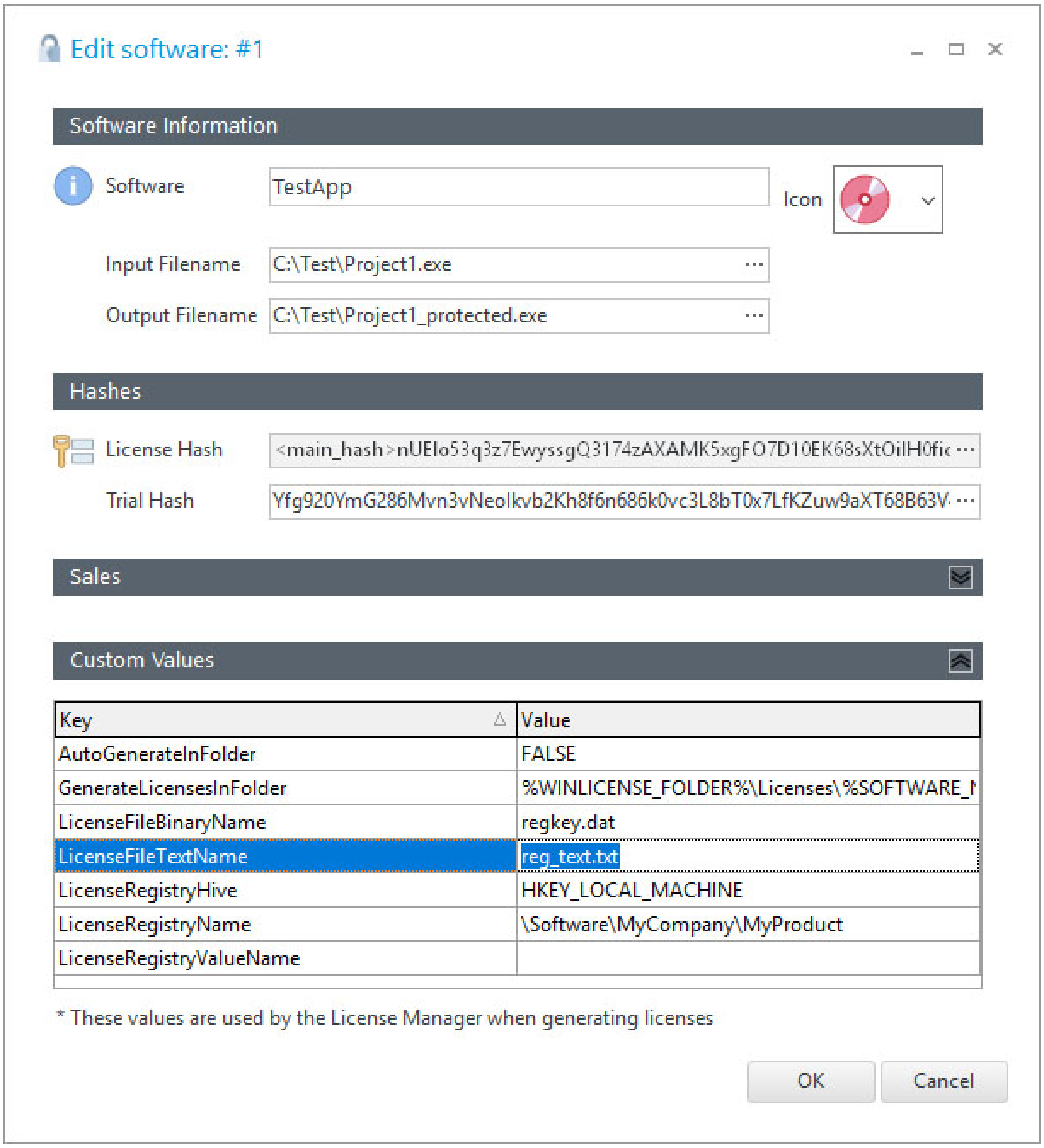
($Product_Name)
($Product_Version)
($Product_Description)
($Product_UnitPrice)
($Product_ShippingFee)
($Product_ServiceFee)
($Product_CustomInfo)
($Product_Notes)
($Customer_Salutation)
($Customer_FirstName)
($Customer_LastName)
($Customer_JobTitle)
($Customer_Organization)
($Customer_Email)
($Customer_Phone)
($Customer_Fax)
($Customer_Address1)
($Customer_Address2)
($Customer_City)

From (name):	TestCompany
From (e-mail):	info@TestCompany.com
CC (e-mail):	
BCC (e-mail):	
Subject:	Initial Tests
Body:	[✓] Attach Registration License; Send as HTML [ ]

```
Dear ($Customer_FirstName),

You can find attached your license for ($Product_Name).

Best Regards,
Test Company
```

In the email body, you can insert specific macros that will be replaced at a later time with the specific value. You just need to double click on the specific macro and it will be inserted in the email body at the current cursor location.

If you are going to attach the generated license file via email, you have to make sure that you have defined the key "**LicenseFileBinaryName**" in the "**Custom Values**" section (in the Software panel). In case that you are attaching a text key (instead of binary file) you have to define the key "**LicenseFileTextName**". Example:



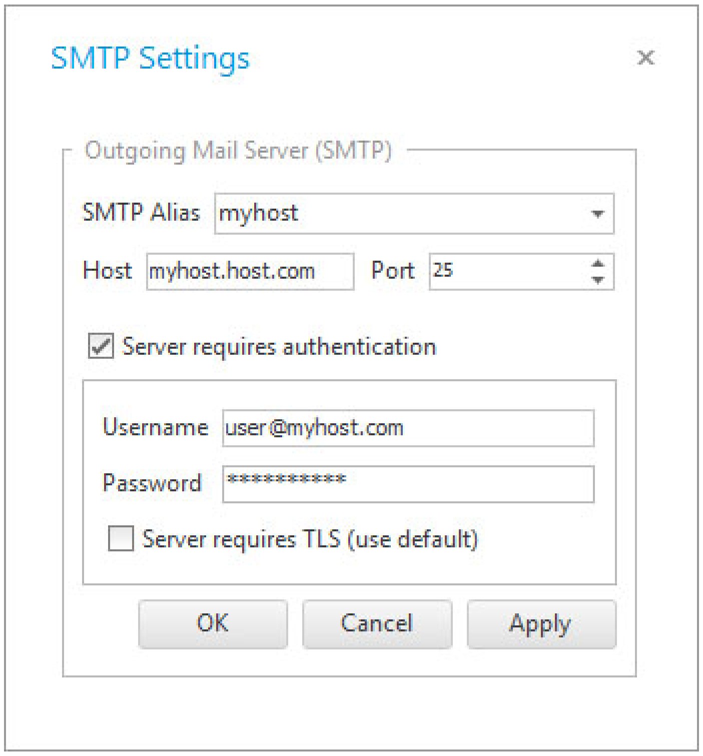
Software	TestApp
Icon	
Input Filename	C:\Test\Project1.exe
Output Filename	C:\Test\Project1_protected.exe
License Hash	<main_hash>nUElo53q3z7EwyssgQ3174zAXAMK5xgFO7D10EK68sXtOilH0fic
Trial Hash	Yfg920YmG286Mvn3vNeolkvb2Kh8f6n686k0vc3L8bT0x7LfKZuw9aXT68B63V

Key	Value
AutoGenerateInFolder	FALSE
GenerateLicensesInFolder	%WINLICENSE_FOLDER%\Licenses\%SOFTWARE_I
LicenseFileBinaryName	regkey.dat
LicenseFileTextName	reg_text.txt
LicenseRegistryHive	HKEY_LOCAL_MACHINE
LicenseRegistryName	\Software\MyCompany\MyProduct
LicenseRegistryValueName	

* These values are used by the License Manager when generating licenses

SMTP Settings

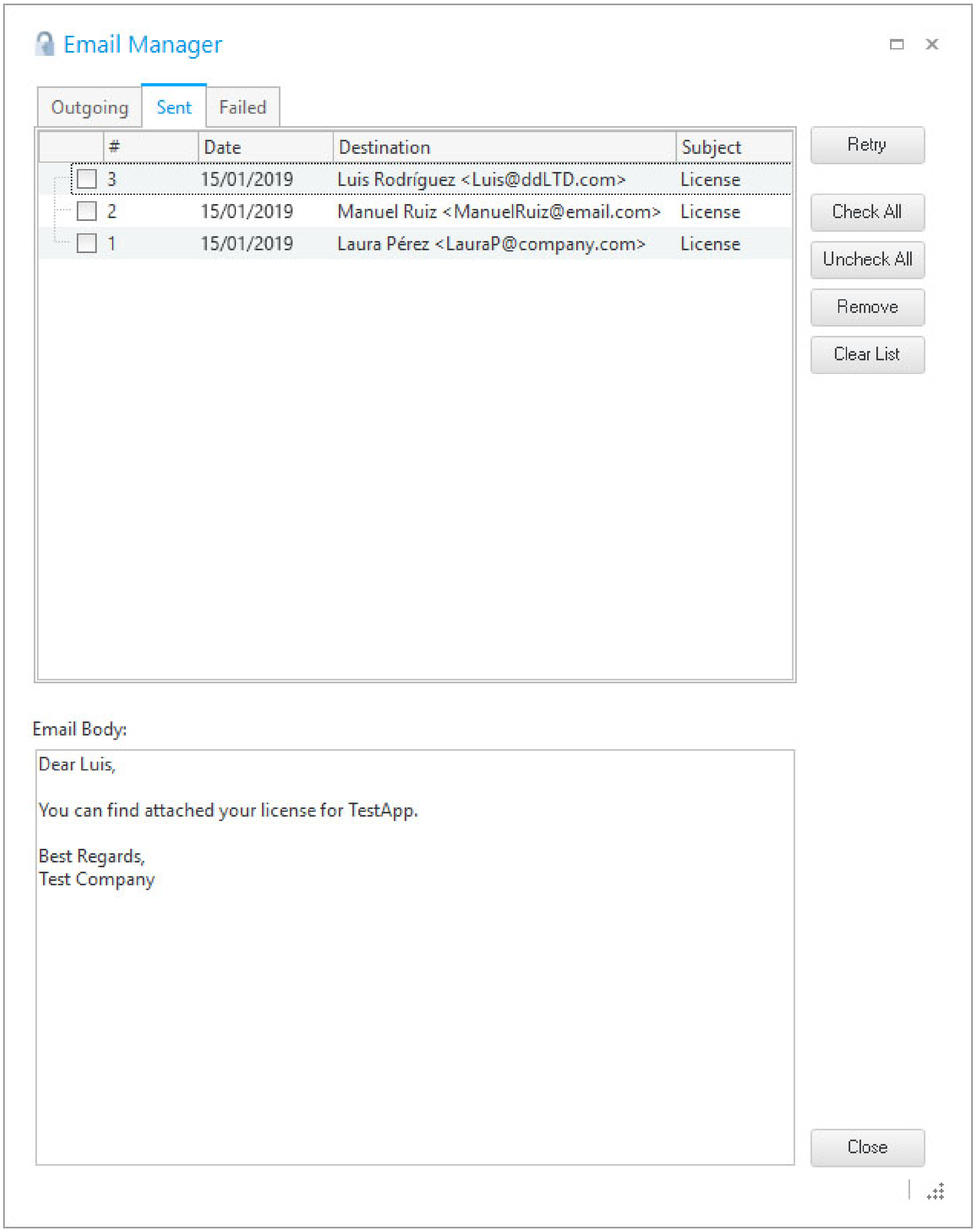
The E-mails Manager can be accessed from the top menu "**E-Mails-->SMTP Settings**". From this form, you can set your SMTP settings in order to be able to send emails to your customers. Make sure that you write the correct values according to your email server.



SMTP Alias: myhost
Host: myhost.host.com
Port: 25
[✓] Server requires authentication
Username: user@myhost.com
Password: ***********
[ ] Server requires TLS (use default)

E-Mail Manager

The E-mails Manager can be accessed from the top menu "**E-Mails-->E-Mails Manager**". This option will display a new form where you can see a panel with of all your emails that have been sent and emails that have being explicitly queued or failed to be sent. You can send and re-send emails from this panel.



#	Date	Destination	Subject
3	15/01/2019	Luis Rodríguez <Luis@ddLTD.com>	License
2	15/01/2019	Manuel Ruiz <ManuelRuiz@email.com>	License
1	15/01/2019	Laura Pérez <LauraP@company.com>	License

Email Body:

Dear Luis,

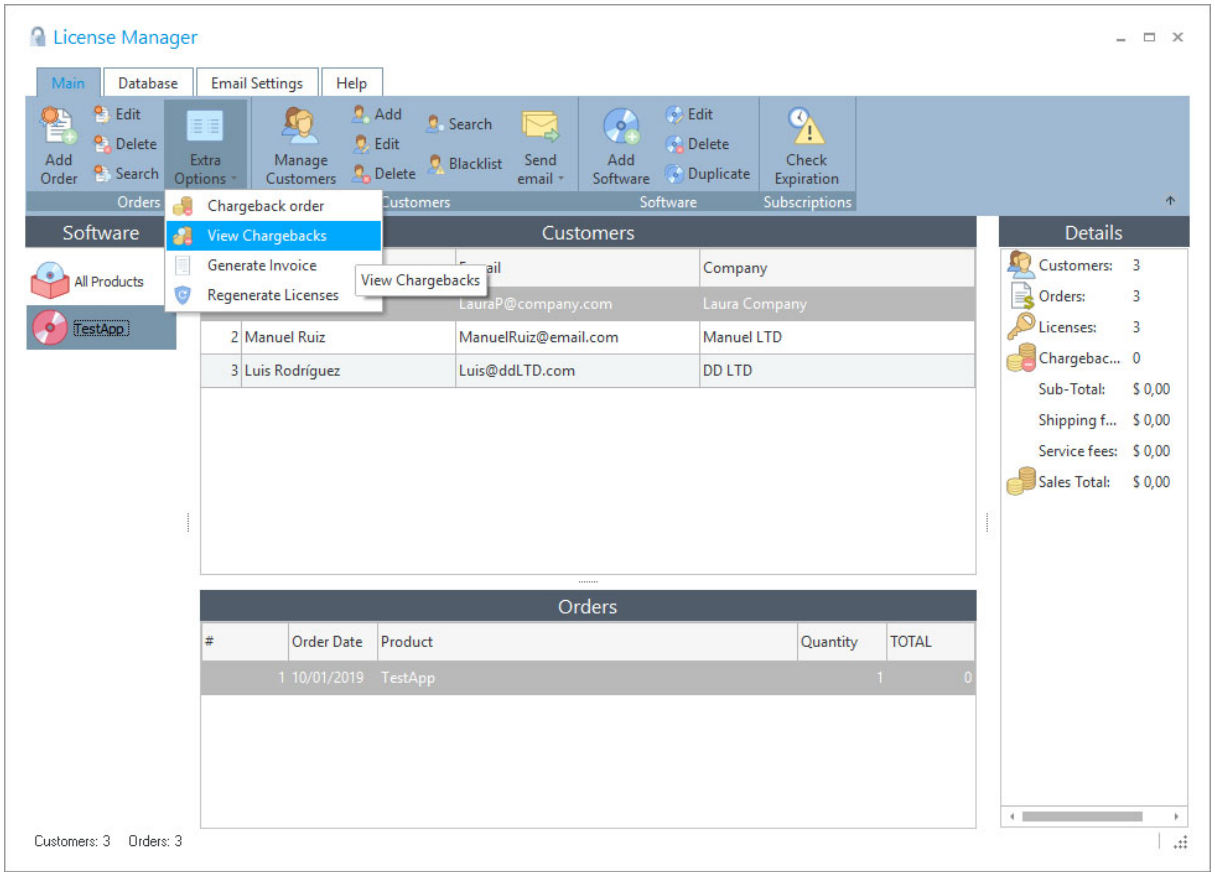
You can find attached your license for TestApp.

Best Regards,
Test Company

2.1.7 Chargebacks

If you receive a chargeback or refund from one of your customers, you can set a specific order as chargeback. At that point the order will be deleted from the database and moved to the Chargebacks table in the database. You can view all your chargebacks from the top menu

"Orders --> View Chargebacks" which will open the Database Manager displaying all your chargebacks using a SQL command.



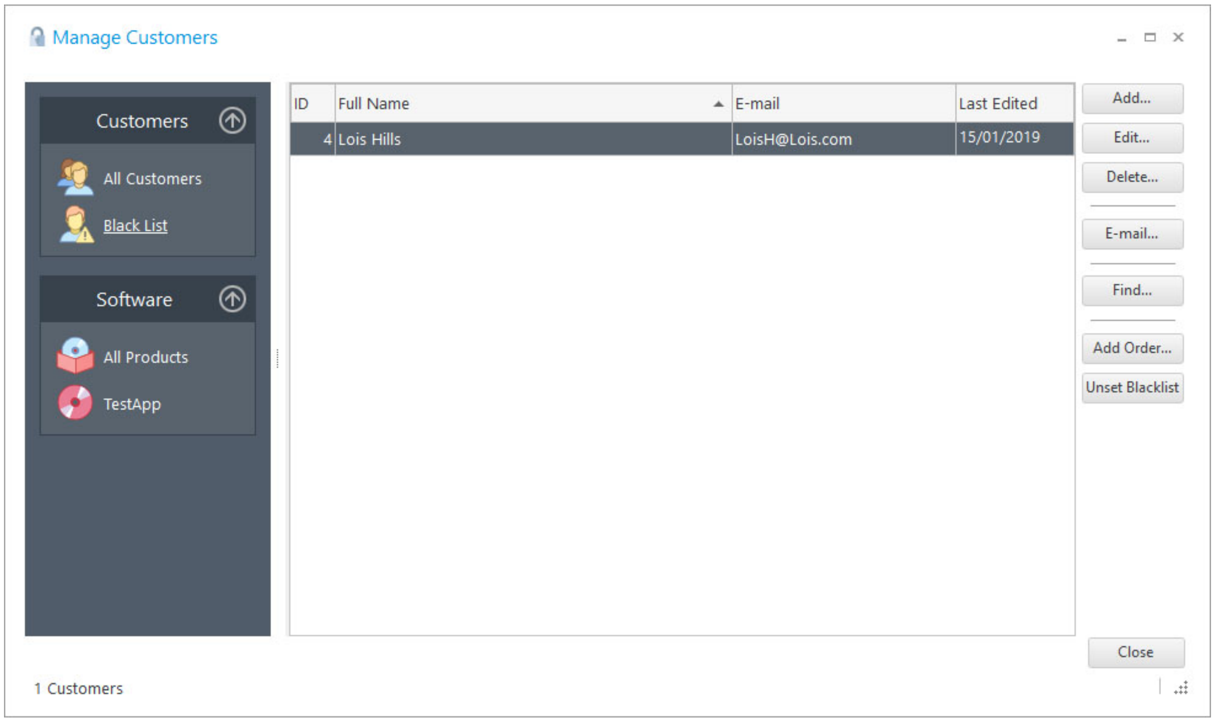
		E ail	Company
		LauraP@company.com	Laura Company
2	Manuel Ruiz	ManuelRuiz@email.com	Manuel LTD
3	Luis Rodríguez	Luis@ddLTD.com	DD LTD

#	Order Date	Product	Quantity	TOTAL
1	10/01/2019	TestApp	1	0

Customers:	3
Orders:	3
Licenses:	3
Chargebac...	0
Sub-Total:	$ 0,00
Shipping f...	$ 0,00
Service fees:	$ 0,00
Sales Total:	$ 0,00

Customers: 3 Orders: 3

If you have received a fraudulent order and you want to put the customer in a black list, you can do it via the top menu "Customers --> Blacklist". When that option is applied all orders belonging to that customer will be deleted from the database and the customer will be marked as "blacklisted". You can see all your blacklisted customers from the "Manage Customers" section.



ID	Full Name	E-mail	Last Edited
4	Lois Hills	LoisH@Lois.com	15/01/2019

2.1.8 Database

You can work with different databases in case that you need it. Each database can be an embedded local database or a server database (MySQL).

When you select a Database, you have to choose if the database is a local database (embedded) or a server database.

If it's a server database, you have to make sure that you have initially created a database in your server and user with all access to the database. After that, you just simply write the login details to the server database and you are done. Please, check with your server administrator about login details to access to the database.

Backing up your database

You should make backups of the database in case of disaster or data loss. There are several ways to backup your database:

1. You can export the database as XML and import it at a later time (from the top menu Database).

2. If you are working with a server database, you can use the backup database facilities that present your server interface.

3. If you are working with an embedded (local) database, you can copy the whole database folder by clicking on the **Backup** button.

2.1.9 Invoice

You can create invoices templates for each specific software. Invoice templates are common DOC/DOCX/RTF documents that you can create with your favorite editor and use special text "variables" that will be replaced with the correct value when the invoice is generated for a specific order.

The following table describes the supported text variables that you can use in your invoice template. Please, refer to the "**Invoice**" sub folder (in your WinLicense installation folder) for an example template that you can have as a reference when you create your invoice templates.

Variable	Description
($DATE)	Current system date
($TIME)	Current system time
($Product_Name)	Product name
($Product_Version)	Product version
($Product_Description)	Product description
($Product_UnitPrice)	Product unite price
($Product_ShippingFee)	Product shipping fee
($Product_ServiceFee)	Product service fee
($Product_CustomInfo)	Product custom info
($Product_Notes)	Product notes
($Customer_Salutation)	Customer salutation
($Customer_FirstName)	Customer first name
($Customer_LastName)	Customer last name
($Customer_JobTitle)	Customer job title
($Customer_Organization)	Customer organization
($Customer_Email)	Customer email
($Customer_Phone)	Customer phone

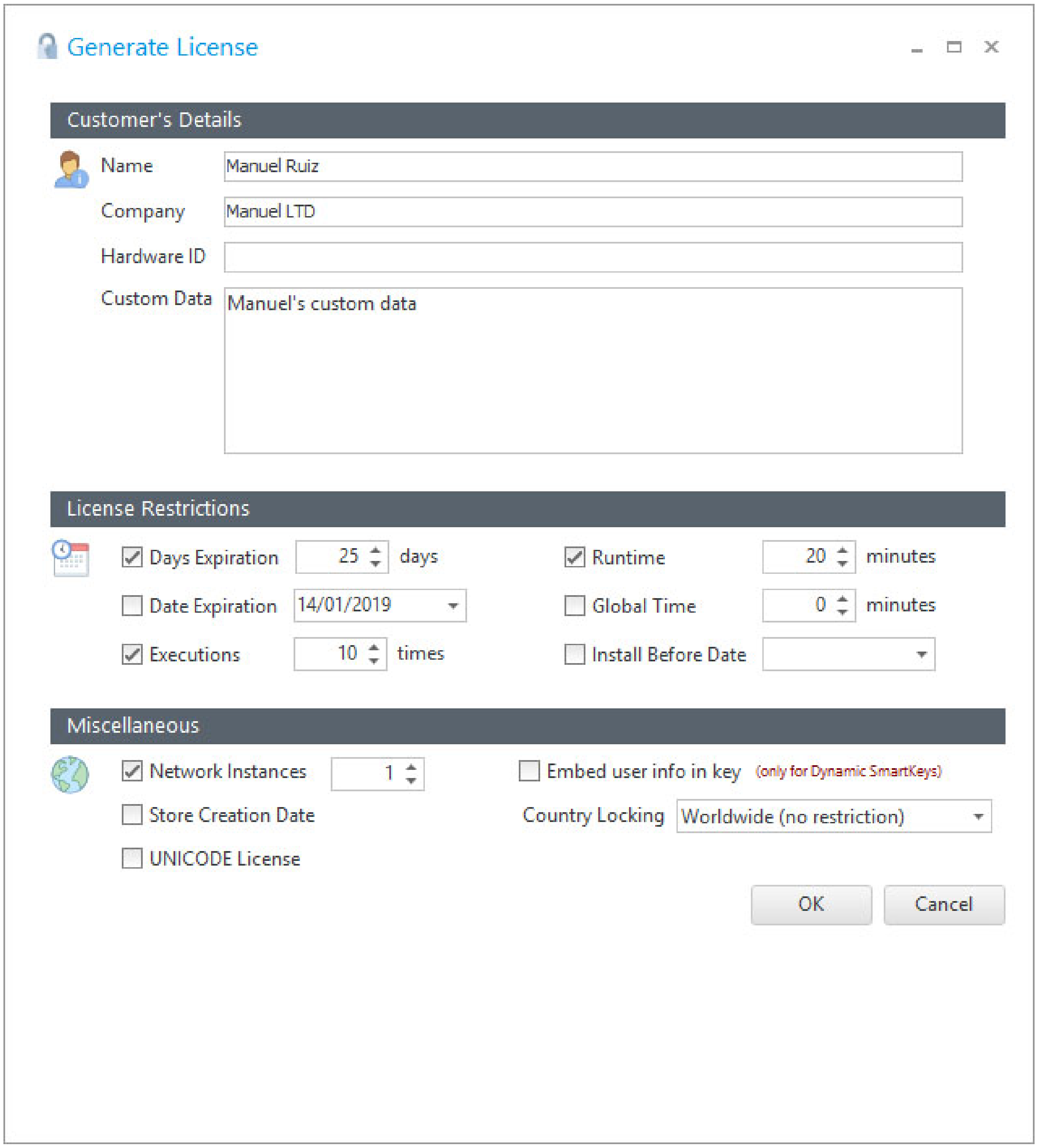
($Customer_Fax)	Customer fax
($Customer_Address1)	Customer address 1
($Customer_Address2)	Customer address 2
($Customer_City)	Customer city
($Customer_Zip)	Customer ZIP
($Customer_State)	Customer state
($Customer_Country)	Customer country
($Customer_Notes)	Customer notes
($Customer_DateEdit)	Customer date edit
($ID_Order)	Order ID
($Order_Date)	Order date
($Order_ExpireDate)	Order expire date
($Order_DeliveryMethod)	Order delivery method
($Order_RegistrationName)	Order registration name
($Order_ShippingAddress)	Order shipping address
($Order_ShippingCity)	Order shipping city
($Order_ShippingState)	Order shipping state
($Order_ShippingZip)	Order shipping ZIP
($Order_ShippingCountry)	Order shipping country
($Order_UnitPrice)	Order unit price
($Order_ShippingFee)	Order shipping fee
($Order_ServiceFee)	Order service fee
($Order_Quantity)	Order quantity
($Order_Total)	Order total
($Order_PaymentMethod)	Order payment method
($Order_Notes)	Order notes
($License_HardwareId)	License Hardware ID (that has been inserted in the generated license)

($License_KeyAsText)	License key as text
($License_UserName)	License user name (that has been inserted in the generated license)
($License_UserCompany)	License user company (that has been inserted in the generated license)
($License_CustomData)	License custom data (that has been inserted in the generated license)
($License_DaysExpiration)	License days expiration (that has been inserted in the generated license)
($License_DateExpiration)	License date expiration (that has been inserted in the generated license)
($License_Executions)	License executions (that has been inserted in the generated license)
($License_RuntimeExecution)	License run-time executions (that has been inserted in the generated license)
($License_GlobalTime)	License global time (that has been inserted in the generated license)
($License_InstallBeforeDate)	License installation before date (that has been inserted in the generated license)
($License_NetworkInstances)	License network instances (that has been inserted in the generated license)
($License_StoreCreationDate)	License store creation date (that has been inserted in the generated license)
($License_EmbedUserInfoInKey)	License embed user info in key (that has been inserted in the generated license)
($License_UnicodeLicense)	License is Unicode (that has been inserted in the generated license)
($License_LockedCountry)	License locked country (that has been inserted in the generated license)
($License_LicenseFormat)	License format

2.2 License Generation

2.2.1 Generating a license

From the Orders panel 374, you can generate a license for your protected applications by clicking on the **Generate** button under the License tab**.** After that, the next form will be displayed:



Name	Manuel Ruiz
Company	Manuel LTD
Hardware ID	
Custom Data	Manuel's custom data

✓ Days Expiration	25 days	✓ Runtime	20 minutes
☐ Date Expiration	14/01/2019	☐ Global Time	0 minutes
✓ Executions	10 times	☐ Install Before Date	

✓ Network Instances	1	☐ Embed user info in key	(only for Dynamic SmartKeys)
☐ Store Creation Date		Country Locking	Worldwide (no restriction)
☐ UNICODE License			

Customer's Details

For each license that you generate, you can insert specific customer's information (Name, Company and Custom Data). By default, WinLicense takes the Customer information from the one that you have generated in the WinLicense database. Notice that you can change the Customer Name in the license even if you have a different customer name in the WinLicense database. The user information, *Name + Company + Custom Data*, can be retrieved in runtime when you call the WinLicense SDK function WLRegGetLicenseInfo 156.

The "Custom Data" field can contain up to 8000 chars of specific information. Some developers want to put special information in the Custom Data field, for example, they encode a value that represent if a special feature in their applications will be active for that license. In runtime, they call the function WLRegGetLicenseInfo 156 and read that value of the "Custom Data" from the license. After parsing the information from the retrieved Custom Data they will enable and or disable specific functionality on their applications.

The **Hardware ID** field allows you to lock a license to a specific customer's machine. In order to fill the Hardware ID field, you first need to know the Hardware ID of your customer's machine. To do so, you can create an external application (or from your main application) and call the WinLicense SDK function WLHardwareGetId 248. After that, the customer will send you the displayed ID and you can fill the **Hardware ID** field for the license that you are going to generate.

Once that you generate a license locked to a specific Hardware ID, each time that your customer launches your protected application, the internal protection code will check if the customer's PC hardware ID matches with the one in the license. If the hardware ID does not match, WinLicense will signal the event **MSG_ID_WRONG_HW_ID** (from the Customized Dialogs 43 panel). In case that the hardware ID matches, your application will start running normally in memory in registered mode.

License Restrictions

WinLicense accepts different type of restrictions for the generated licenses. You can create licenses with no expiration restrictions at all (so the license will work forever) or add one or more different expiration restrictions to a single license, so your license can expire for example by number of days or by number of executions (the one that happens first)

- **Days Expirations**: This is the number of days that your license can run since it was initially used. When your customer runs your application with a license with days expiration, WinLicense will keep track of the day that the license was first used. After that, the license will expire after the given number of days. When the license expires by days, WinLicense will signal the event **MSG_ID_LICENSE_DAYS_EXPIRED** (from the Customized Dialogs 43 panel)

- **Date Expiration**: You can set a specific expiration date for your license. The license will expire when the expiration date arrives, even if the user has no launched the application before that date. When the license expires by date, WinLicense will signal the event **MSG_ID_LICENSE_DATE_EXPIRED** (from the Customized Dialogs 43 panel)

- **Executions**: You can restrict a license to be only used a specific number of times on a machine. Each time that your application is launched, the internal execution counter will add one execution. When there are no more executions left, WinLicense will signal the event **MSG_ID_LICENSE_EXECUTIONS_EXPIRED** (from the Customized Dialogs 43 panel)

- **Runtime**: You can restrict your license to be only used a specific amount of minutes in memory. Each time that the application is launched, the runtime expiration starts again from zero till it reaches the specific limit in the license. When your application has been running in memory during the specific runtime limit, WinLicense will signal the event **MSG_ID_LICENSE_RUNTIME_EXPIRED** (from the Customized Dialogs 43 panel)

- **Global Time**: You can set the maximum time (minutes) that a license can be used on a machine. For example, if you set 180 minutes as global time expiration, your customer can launch your application multiple times but WinLicense keeps a counter of the number of minutes that your application has been in memory. Once that the customer has been running your application during more than 180 minutes (no matter if he just launched one time and consumed the 180 minutes or it took him 3 months to reach the 180 minutes) your license will be expired. When the global time is expired, WinLicense will signal the event **MSG_ID_LICENSE_GLOBALTIME_EXPIRED** (from the Customized Dialogs 43 panel)

- **Install Before Date**: You can force a license to be installed on a machine before a specific date. If your customer installs the license after the specific date, WinLicense will signal the event **MSG_ID_LICENSE_INSTALLATION_TIME_EXCEEDED** (from the Customized Dialogs 43 panel)

Miscellaneous

Apart from time restrictions, you can also set other special options in your WinLicense licenses.

- **Store Create Date**: This option inserts extra information in the generated license to store the date that the license was generated. You can retrieve the license creation date by calling the WinLicense SDK function WLRegLicenseCreationDate 164

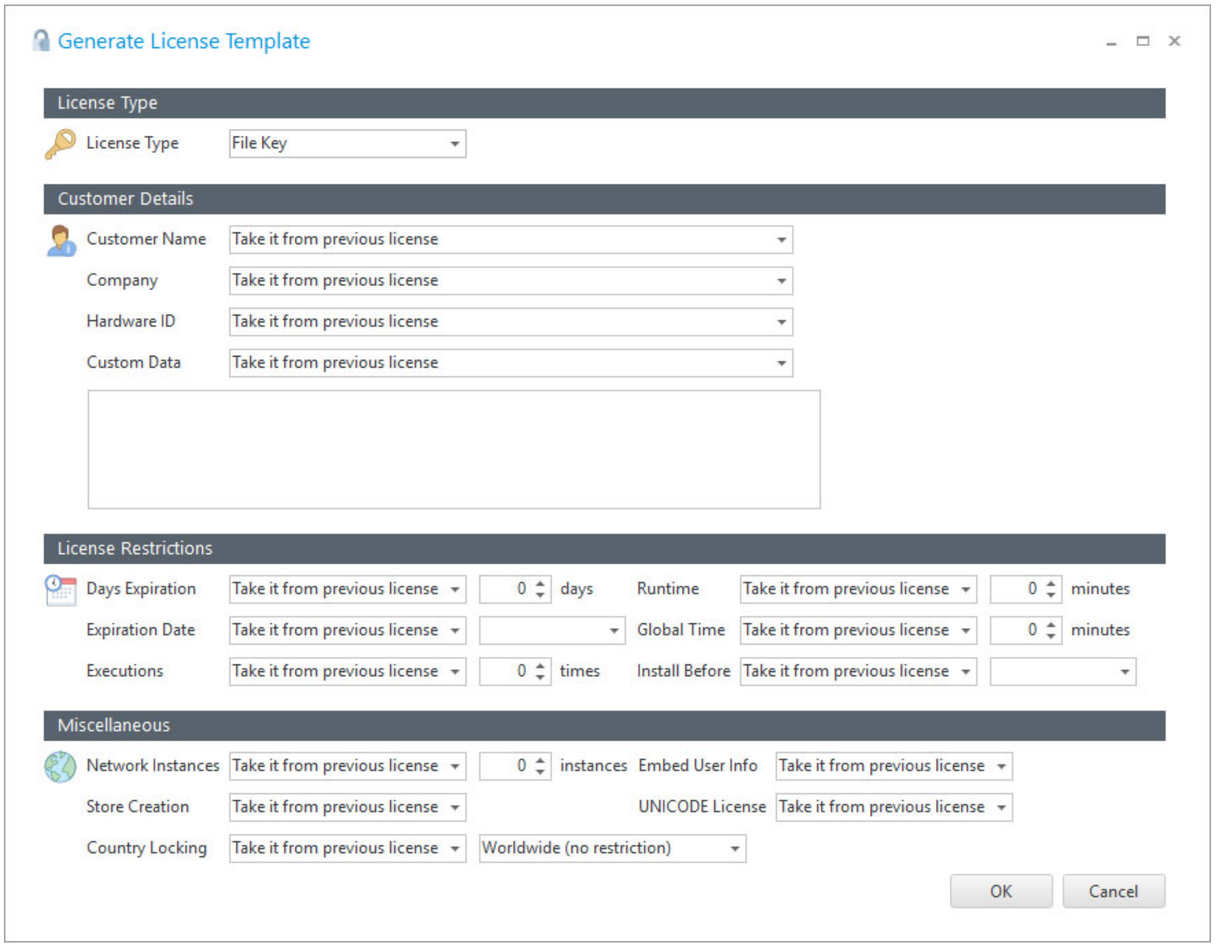
- **UNICODE License**: If you are going to work with UNICODE licenses you should enable this option. When working with SmartKeys, you have to make sure that if you work with Unicode licenses, you call the WinLicense SDK SmartKey functions in their Unicode form (WLRegSmartKeyCheckW 180, WLRegSmartKeyInstallToFileW 183, WLRegSmartKeyInstallToRegistryW 191)

- **Embed User Info in Key**: This option is only for Dynamic SmartActivate keys 36. If you select that option, the SmartKey serial number will contain the user information as part of the serial number. When you call the SmartKey functions (WLRegSmartKeyCheck 178, WLRegSmartKeyInstallToFile 182, WLRegSmartKeyInstallToRegistry 189), you have to pass NULL for the user information parameters. Example: `WLRegSmartKeyCheck(NULL, NULL, NULL, pSmartKey.` Notice that when the user information is embedded inside the dynamic SmartKey, the generated serial number will be bigger as it contains the user information encoded as part of the serial number.

- **Country Locking**: This option is used to create a restricted license that can only be used in a specific country. Notice that the country is determined by the Windows Language settings and it's not a real proof that the user is really running your application from a specific country. If the current language does not match the specific country inside the license, WinLicense will signal the event **MSG_ID_LICENSE_COUNTRY_ERROR** (from the Customized Dialogs 43 panel)

- **Network Instances**: This option is used to limit the number of instances inside a network. Please, refer to the Network Instances (Floating licenses) 101 section for more information.

2.2.2 Regenerating all licenses for a Software


Generate License Template

License Type

License Type	File Key						

Customer Details

Customer Name	Take it from previous license
Company	Take it from previous license
Hardware ID	Take it from previous license
Custom Data	Take it from previous license

License Restrictions

Days Expiration	Take it from previous license	0	days	Runtime	Take it from previous license	0	minutes
Expiration Date	Take it from previous license			Global Time	Take it from previous license	0	minutes
Executions	Take it from previous license	0	times	Install Before	Take it from previous license		

Miscellaneous

Network Instances	Take it from previous license	0	instances	Embed User Info	Take it from previous license
Store Creation	Take it from previous license			UNICODE License	Take it from previous license
Country Locking	Take it from previous license	Worldwide (no restriction)			

Sometimes you might need to regenerate all licenses for a specific software, for example, you want to send a new license to all your customers with a new expiration date or you want to re-generate all your licenses after changing your Software License Hash 363. You can re-generate all licenses for all orders made on a specific software. To do so just go to the top menu **Main** and select **Extra Options --> Regenerate Licenses**. After that a new form will be displayed which allows you to select the restrictions and/or modifications for all licenses that are going to be generated.

In the displayed form, you can select if you want to keep previous license settings/restrictions on the new generated licenses. For example, suppose that you created a license for "Peter Koyl" with 30 days restrictions and another license for "Karol K" with 5 executions expiration. If you select **Take it from previous license** on the **Days Expiration** and **Executions** field, the new licenses will also be restricted for 30 days for "Peter Koyl" and the other one restricted for 5 executions for "Karol K".

If you want that all new licenses will just contain a specific expiration date and remove all previous license restrictions (that were generated on the previous license) you should set all license restrictions to **Disabled** and just set the **Date Expiration** field to "**Set it to -->**" and insert the new expiration date for all new generated licenses.