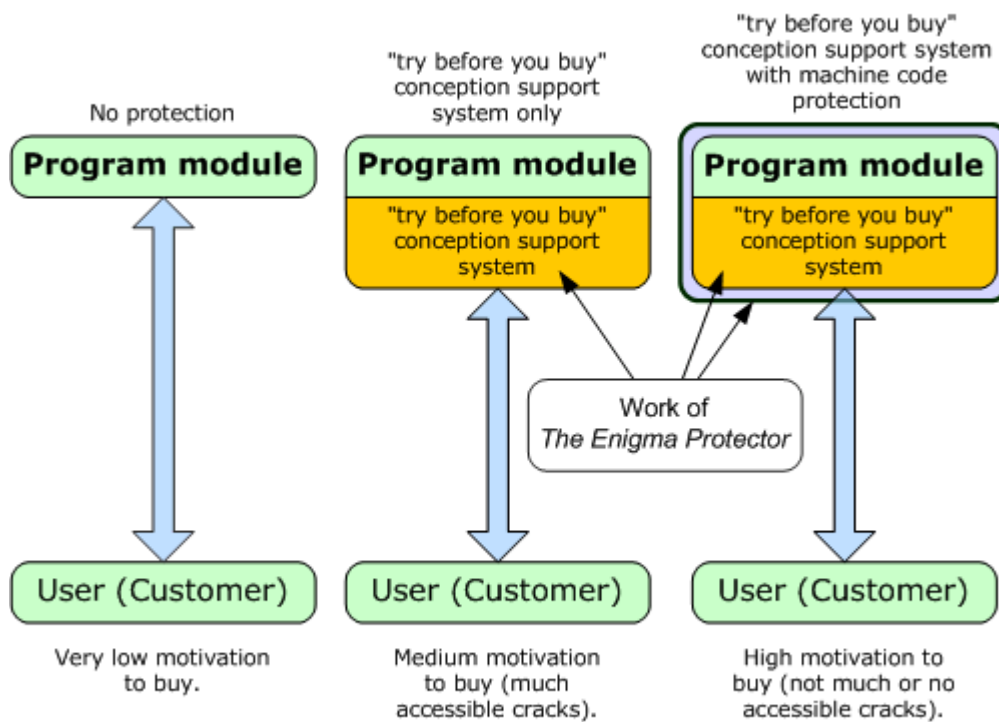


The Enigma Protector

The Enigma Protector is a powerful tool designed for complex protection of program modules. Program modules include the following types of objects:

- | Win32 Portable Executable file (*.exe);
- | Windows Screen saver files (*.scr);
- | Dynamic Link Libraries (*.dll);
- | 32 bit ActiveX control files (*.ocx);
- | .NET executables (*.exe).

In this context the term "*protection*" means realization of two major ideas. The first one is "try before you buy" concept support system (the mentioned concept is the main principle of the shareware marketing method). And the second one is protection of program module machine code from analysis and cracking. As it can be seen, the protection is realized in different, but nevertheless greatly interrelated aspects which are protection of the developer's economic interests and technical protection of a software product. Later we will consider these ideas in more detail.



The Enigma Protection and motivation for buying

"Try before you buy" conception support system

Almost every commercially distributed software product has a customer-key protection system. The software developers have to build various enciphering algorithms into their programs. In many cases these algorithms have low anti-cracking and anti-direct-enumeration protection rate. Suggested "try before you buy" concept support system provides functionality which is necessary to work with registration keys. It gives easy ways of interaction between the software developer and the user (customer). Functions of unique keys generation (within a certain project scope) have a high protection rate. The Enigma Protector uses enciphering algorithms which are congruous to RSA algorithm with the key of 512 bit length. This makes it impossible for a cracker to create spurious registration keys. Functionality of the program module trial limitation is also provided. The user who works with the protected program has an ability to try its operational capability within some period of time. At the end of this period, the program module can no longer work normally. To restore this ability, it is necessary to perform the ordering procedure (which usually requires payment).

Protection of program module machine code

During the distribution of their own program product, every shareware developer faces the problem of cracking. Herewith serial numbers, spurious key generators and so on get a wide access. The effect of this "try before you buy" concept is invalidated and the shareware marketing method loses its efficiency. For the developer this reveals itself in reduction of amount of monthly sales and loss of profit. The suggested protection of machine code includes creation of series of considerable hardships on the way of persons (crackers) who try to perform an unauthorized code analysis and modifications for the purpose of illegal use. Machine code compression and enciphering are supported. It is possible to make a considerable reduction of the program module size without the loss of its working capacity.

To start working with The Enigma Protector read the following sections:

- | [Getting Started](#)
- | [Tutorials](#)

Managing of licenses

The Enigma Protector includes several unique tools for managing licenses:

- | [License manager](#) serves for storing licenses generated for the registered users. License manager allows to create customer records and generate licenses. It is a very useful, user-friendly and simple manager!
- | [Mailer](#) serves for automatic generation and sending of emails to the users stored in the database. For any software developer sending notification emails to the registered users is a common thing. For example, if the newest version of the software has been released or a new registration scheme has been implemented, the Mailer provides you with a very simple way of automatic email generation and sending.

What can The Enigma Protector do?

The Enigma Protector has a hard scheme of registration keys generation:

- | comfortable interface for creating and verifying registration keys. You do not need to look for any safe decisions on how to generate registration keys for customers. The Enigma Protector helps you to create keys with a very safe algorithm like RSA with up to 512-bits key length!
- | special Enigma API. Enigma API is a set of special functions to communicate between the module and the Enigma loader. You are able to get full information about registration keys, current trial parameters, etc.
- | hardware locking of registration keys. This perfect feature helps you to generate registration keys for a particular computer only! The registration key generated with the hardware locking function enables work only on one PC you have chosen.
- | time limited registration keys. If you need to limit the time of usage of a registered version of the module, you will be able to do it, just create a time-limited registration key!

The Enigma Protector has a wide range of features to limit time of module usage:

- | executions, days, date, time limitations. The main idea of shareware modules is "try before you buy". The customer should see how the application works and what features it has, let's show these features but do not forget to limit usage time to increase motivation to buy application.
- | system clock control. This feature is used to control system clock reversing. It helps you to avoid dishonest customers.

The Enigma Protector has a lot of features to make your software resistant to cracking:

- | anti-debugger tricks. Debuggers are the special tools that allow you to reverse source machine codes of the executed module. All reverse engineers use these tools to understand how your module works or how the module protection works. Using this feature helps to avoid simultaneous execution of the protected module with debug tools.
- | control sum checking. Control sum is special data which helps to understand if the data region is modified or not. Every crack (removing of protection etc.) needs to modify some of machine code region. The Enigma Protector is able to check if the sources are modified and an alert is made. The Enigma Protector checks not only machine codes of the protected module, but also own sources!
- | set startup password. Sometimes you need to limit count of users who are using protected module to a particular group, startup password feature is the most safe decision.
- | any additional features to check number of simultaneously executed copies of the protected module, file name of the module, disk type where the module is executed.

- | checkup of external files. If your application package contains any other files except the main protected executable module you may use this feature to control these files against modifications.
- | checkup of executed processes. This feature is used for checking if some files/processes are executed. It is performed by checking of module file name, process windows text or class. This may help to avoid execution of protected module if any debuggers/screencaptures/monitores are executed.
- | checkup of installed services.
- | checkup of Windows version.
- | checkup of Virtual Machines. If the file is executed under Virtual Machine (VMWare, Virtual PC etc) the execution is terminated.
- | hard modifications of import table of the executable. Nobody will know what import libraries your module uses.

The Enigma Protector has features to help the programmer to add beautiful things into existing module without writing any additional sources strings:

- | splash screen. Add splash screen to the module startup. Choose your own picture to be shown when the module starts.

Order

To order a license for The Enigma Protector, please visit www.enigmaprotector.com. Prices for different types of licenses are given in the following table.

Name of product	Price of Single Developer License	Price of Company license
The Enigma Protector Professional	149 USD	299 USD

This license provides you with free updates within 9 months.

Money transfer services

Purchase of The Enigma Protector can be performed through the following money transfer services: ShareIt! (www.shareit.com), Western Union (www.westernunion.com), WebMoney (www.wmtransfer.com). For purchasing with the Western Union and WebMoney, please, contact to our support team for more information support@enigmaprotector.com.

About ShareIt!

The ShareIt! supports the following types of purchases (this information was taken from ShareIt! official site):

Credit Card

- | We accept Visa, MasterCard, American Express, Diners Club, and JCB.
- | When you pay by credit card, your order will be processed immediately. Postal mail shipments are initiated immediately. Products available electronically are generally ready for download immediately, or no more than 48 hours after you place your order.

Solo/Switch/Maestro (only if issued in UK)

- | We accept debit cards from Switch/Maestro and Solo (UK).
- | When you pay by debit card, your order will be processed immediately. Postal mail shipments are initiated immediately. Products available electronically are generally ready for download immediately, or no more than 48 hours after you place your order.

Bank/Wire Transfer

- | To help minimize fees for wire transfers to foreign countries, you may send bank transfers to one of our accounts in Germany, France, Great Britain, Finland or Japan.
- | You will receive the account information via e-mail after your order is processed.
- | You will receive your product once the payment is received.

PayPal

- | PayPal payments for orders are currently accepted only in USD, EUR, CAD, GBP and JPY.
- | A valid PayPal account is not required to use this payment method; you can set up an account when you are directed to PayPal's Web site.
- | If you choose PayPal as your payment option, you will be automatically redirected to PayPal's Web site. You can then log in to PayPal as usual or set up the PayPal account to pay for your order.
- | During the payment process, all information will be exchanged in encrypted format exclusively between you as the account holder and PayPal's Web site.
- | After the transaction is successfully completed, you will be returned to share-it!'s order process pages.
- | Typically, product delivery will be initiated immediately.

Check

- | We accept all personal, business, and Cashier's checks. Please note that personal checks may be held for up to 10 business days.
- | Please note that postal money orders must be in US dollar currency and drawn on US Bank. Please send those postal money orders to our US-Office:

ShareIt! Inc.
9625 West 76th Street, Suite 150
Eden Prairie, MN 55344

- | You will receive detailed information by e-mail after your order is processed.
- | Products will be shipped once payment is received.

Cash

- | You can also send us cash by mail. You will receive notification of our postal address by e-mail after your order is submitted.
- | Your order will be shipped as soon as the payment is received.

Support

If you have any technical problems with The Enigma Protector or need a special feature to be included in the next release, please feel free to contact us at support@enigmaprotector.com.

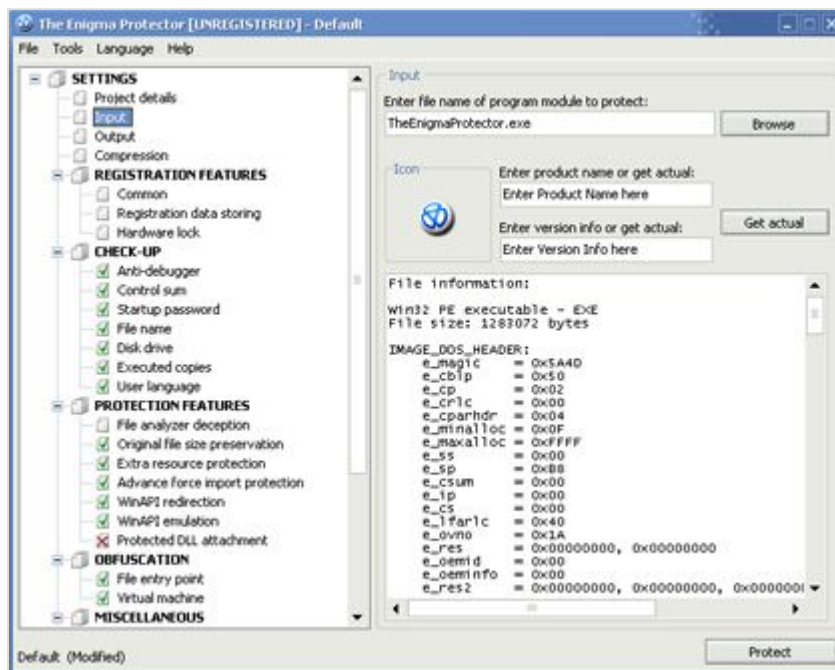
In case of technical problems with The Enigma Protector, make sure that you are using the latest version available. You can download the latest version of The Enigma Protector from www.enigmaprotector.com.

Please, try to find the requested information in the FAQ section or at our support forum!

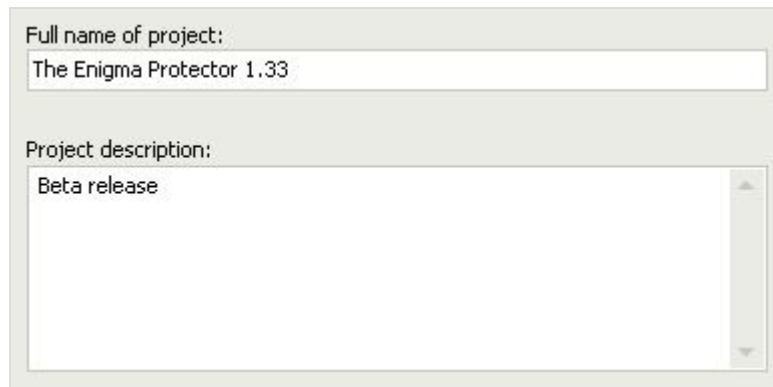
Program Overview

This section unites description of the main features of The Enigma Protector categorized into the following subsections :

- | Project details
- | Input
- | Advance Input
- | Output
- | Registration Features
- | Check-Up
- | Protection Features
- | Virtual Box
- | Virtual Machine
- | Miscellaneous
- | Trial Control



Project details




A screenshot of a web form titled "Project details". The form has a light gray border and contains two input fields. The first field is labeled "Full name of project:" and contains the text "The Enigma Protector 1.33". The second field is labeled "Project description:" and contains the text "Beta release". The description field has a vertical scrollbar on its right side.

Enter the name and description of the project. These fields are used only to provide some information to let you know what this project is about. You are able to change/modify these data at any time without losing module workability.

Input

Enter file name of program module to protect:
TheEnigmaProtector.exe

Icon


Enter product name or get actual:
Enter Product Name here

Enter version info or get actual:
Enter Version Info here

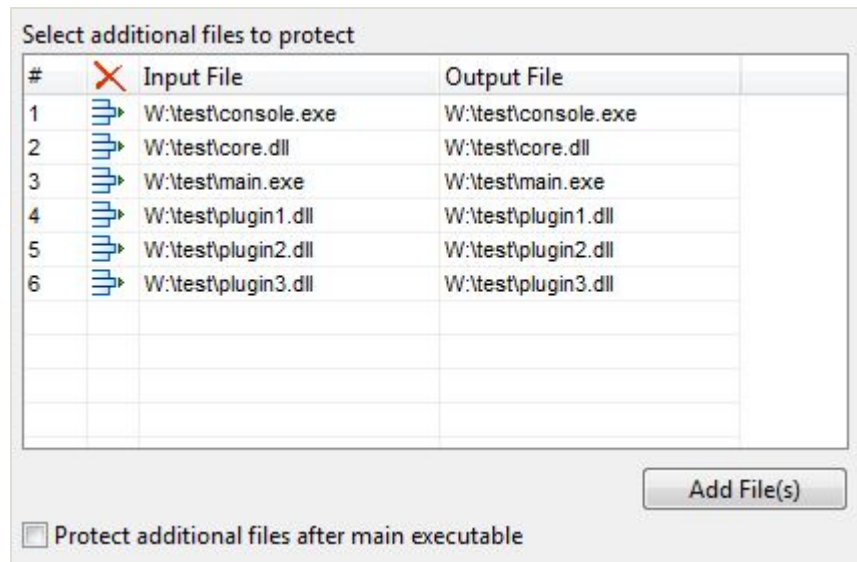
File information:
Win32 PE executable - EXE
File size: 1283072 bytes

IMAGE_DOS_HEADER:
e_magic = 0x5A4D
e_cblp = 0x50
e_cp = 0x02
e_crlc = 0x00
e_cparhdr = 0x04
e_minalloc = 0x0F
e_maxalloc = 0xFFFF
e_ss = 0x00
e_sp = 0xB8
e_csum = 0x00
e_ip = 0x00
e_cs = 0x00
e_lfarlc = 0x40
e_ovno = 0x1A
e_res = 0x00000000, 0x00000000
e_oemid = 0x00
e_oeminfo = 0x00
e_res2 = 0x00000000, 0x00000000, 0x00000000

This topic includes the most important fields which should be specified.

- | *Enter file name of the program module to be protected* - the file name which needs to be protected.
- | *Icon of the program module* - the icon of the module you have selected.
- | *Enter product name* - the module product name. Warning: this information is used for generation of some protection parameters, don't change this field for the same module.
- | *Enter version info* - module version information. It is used to work with Trial Control functions. Do change product version field for every module version.
- | *Get Actual* - reads the module product name and version from resources.
- | *File information* - special file information, contains the type and size of the file, main values from the header of 32-bits PE Windows executable.

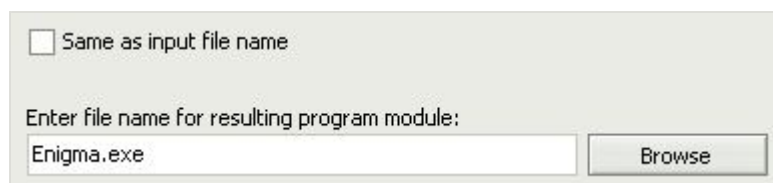
Advance Input



Allows you to select additional files that will also be protected with the current project. Click "Add File(s)" button and select the necessary additional files. To delete a file from the list, click ☰▶ button. To select another output file name, activate the necessary row and click on the "Output File" column.

Protect additional files after the main executable - if selected, then the main executable (the input file selected in Input panel) will be protected before the additional one selected in the list. Otherwise, the additional files will be protected first and the main executable will be at the end of protection.

Output



The image shows a dialog box with a light gray background. At the top, there is a checkbox labeled "Same as input file name" which is currently unchecked. Below this, the text "Enter file name for resulting program module:" is displayed. Underneath the text is a text input field containing the text "Enigma.exe". To the right of the input field is a button labeled "Browse".

Enter the file name of the resulting protected file. If you chose the same input and output file names, the original file will be copied in the same folder with the .bak extension.

Same as input file name - turn this check on to set output file name the same with the original one. The original file name is defined in the Input panel.

Do not create backup (.bak) files - if this option is enabled, no backup file will be created if input and output files are the same. This option also affects the files from [Advance Input](#) panel.

Registration Features

The features described in this section include setting project parameters that are related to storing and using registration keys. To create registration keys, see [Keys Generator](#). The examples of working with registration keys can be viewed in [Enigma API Description](#).

Follow the links below to get more details.

- | [Common](#)
- | [Key Properties](#)
- | [Registration Data Storing](#)
- | [Hardware Lock](#)
- | [Encrypt with Hardware ID](#)
- | [Registration Dialog](#)
- | [Key Expiration Reminder](#)

The screenshot shows a configuration window for registration keys. It has several sections:

- UNICODE Registration Scheme
- Allow execution only if registered
 - Encrypt application with Encryption Constant (please take a look at the manual before using this option)
- Information for Custom Keys Generator**
 - Encryption Constant: 2113444489
 - Private Key: 020MLJ7XTRVLBDNUVMKGUFHCU8XBGMHH\
 - Public Key: 0201B810DA4A1ADD4351378790A981385330CI
- Registration keys type**
 - Registration key safety/length: ~ RSA 1024 bits
 - Registration key output base: Base 32
- Example of the registration key:
 - Mode: 1024
 - Base: 32
 - Aproximate minimum size: 53 symbols
 - Registration key: F7R89KT7ZMC4SVBPE4W2XQ75ZSGPH4L2MUMBCMxefgvaap6CPNSP3

These features are related to different options for registration keys usage.

UNICODE Registration Scheme - enables UNICODE support for the registration system. It means that:

- | [Keys Generator](#) works with unicode support;
- | [Keys Generator of License Manager](#) works with unicode support;
- | [Enigma API](#) with unicode support (functions with the W prefix) should be used for custom checking/saving and loading of registration information;
- | [CGI Keys Generators](#) should be called with unicode support (functions with the W prefix);
- | custom keys generators that use [keygen.dll](#) have to call unicode support functions (functions with the W prefix).

Please note:

- | if you change this feature in the existing project, all previously generated registration keys will become invalid;
- | ANSI functions of checking/saving and loading registration information will fail.

Allow execution only if registered - allow execution of the protected module only if it is registered.

Encrypt application with Encryption Constant - it encrypts the application with the Encryption Constant. The Encryption Constant is a unique value that is stored in the registration key. It means that if the application is protected with this option, it is impossible to run/unpack the application without valid registration keys. This feature has one common restriction: the application should be unlocked with registration keys generated with at least one crypted section enabled. If the registration key does not have any sections unlocked, the application will fail to be executed. If you try to run the application protected with this option and registered with the key that does not have any crypted section enabled, the execution will depend on the options selected on [Checkup-Control Sum](#) panel.

That is, if the control sum is enabled, a message will be shown or the execution will be terminated silently.

Registration keys mode safety/length

Select the mode of registration keys. Available modes: ~ RSA 512/768/1024/2048/3072/4096. Keys with a high count of bits are longer than the ones with a low count, but they are much stronger. Usually, a 512-bit protection is enough.

NOTE: if you change this option, all previously generated keys will be invalid.

Registration keys base

Select the output base of registration keys.

NOTE: if you change this option, all previously generated keys will be invalid.

See [Creating Keys](#) for more information.

Key Properties

- Allow only hardware locked keys
- Allow only keys with expiration date
- Allow only keys with Register After date
- Allow only keys with Register Before date
- Allow only executions limited keys
- Allow only days limited keys
- Allow only run-time limited keys
- Allow only global time limited keys
- Allow only country locked keys

The features from this page allow you to limit the usage of different types of registration keys. For example, if you have enabled the "Allow only hardware locked keys" option, the application will apply only the registration keys that have been generated with the hardware I., If the registration key was generated without a hardware ID, it will be deemed invalid by the application even if it is valid.

Such features are also important for avoiding illegal usage of the registration keys (that were probably stolen or illegally generated) if you are planning to constantly use any of these limitations for all the registration keys.

Allow only hardware locked keys - allows only hardware locked registration keys.

Allow only time limited keys - allows only registration keys with the Expiration Key Date.

Allow only keys with Register After date - allows only registration keys with the Register After date

Allow only keys with Register Before date - allows only registration keys with the Register Before date

Allow only executions limited keys - allows only registration keys that are limited by the number of executions

Allow only days limited keys - allows only registration keys that are limited by the number of days

Allow only run-time limited keys - allows only registration keys that have a run-time limit

Allow only global time limited keys - allows only registration keys that have a Global Time limit

Allow only country locked keys - allows only registration keys that are limited to a particular country

Registration Data Storage

Common

Encrypt Registration Information

Disable copying of the registration information to another PC (the registration information will be encrypted with the user's Hardware ID)

Select place where registration information will be stored:

Use system registry

Use file system

Use both techniques

System Registry

Registry Root: HKEY_CURRENT_USER

Relative path in registry (e.g.: SOFTWARE\\YourCompany): \\SOFTWARE\\EnigmaDevelopers\\

File System

Base folder in file system: %My Documents FOLDER%

Relative path in file system (e.g.: \\YourCompany\\key.dat): \\notepad\\secret.dat

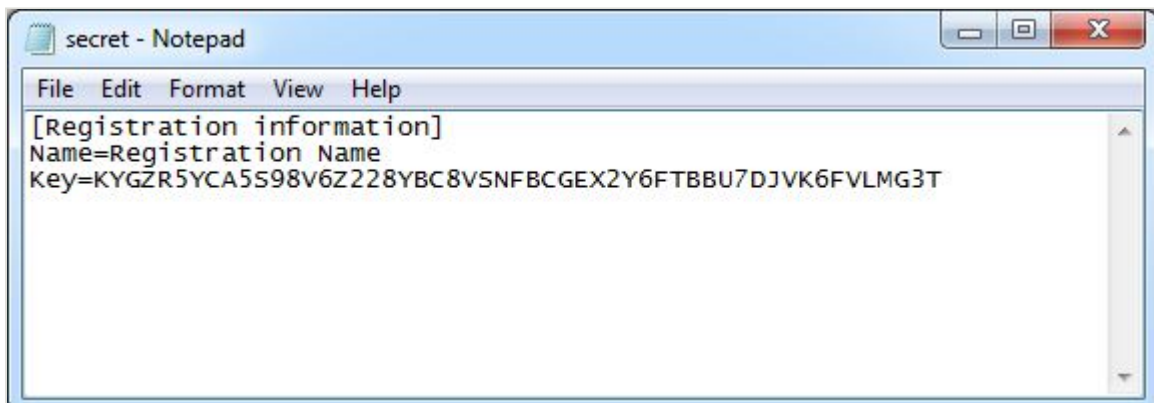
Set File Attributes

This section describes possibilities to select the location where registration information will be stored on the user's computer. Registration information refers to the couple of strings - the registration name and registration key. Registration information is stored in the user's system without being encrypted. This means that it can be accessible for viewing. If you do not use the features of The Enigma Protector for the license creation (for registration keys generation), the data entered in those fields will be ignored by The Enigma Protector.

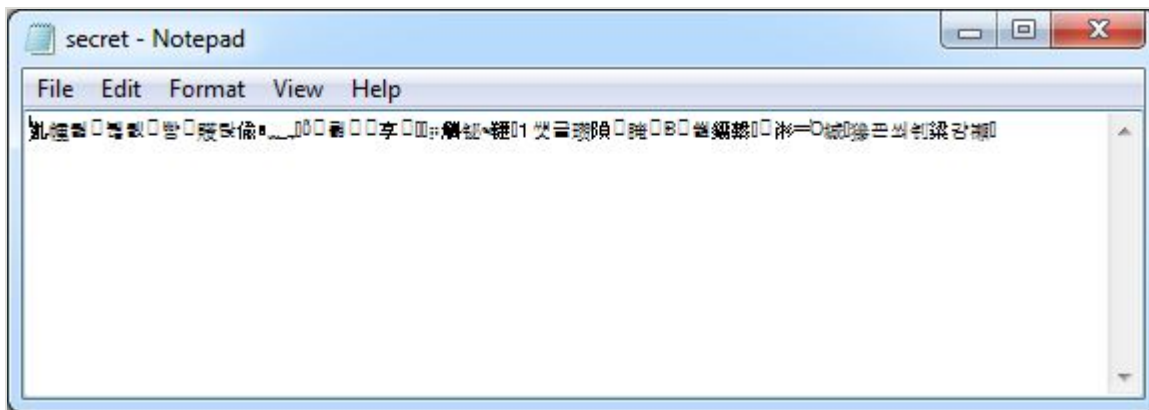
The Enigma Protector supports two types of the registration data storage - the Windows registry and an external file. You can use each type separately or both techniques simultaneously.

Encrypt Registration Information - allows you to encrypt registration information. If the option is disabled, registration information (the name and key) will be stored in a registry/file as they are.

For example, the file with the registration information, without the Encrypt Registration Information feature, will look in the following way:



with the Encrypt Registration Information feature



Disable copy of the registration information to another PC (the registration information will be encrypted with the user's Hardware ID) - this option allows you to encrypt registration information with the user's hardware ID as well. If the registration information is encrypted with the hardware ID, the registry item or file with registration information will work and will be valid only on the PC where it was created. If the user attempts to copy this registry item or file to another PC, registration will fail on this PC.

Please note:

- | This feature is incompatible with the "Allow Hardware Changes" from [Hardware Lock](#) panel. This means that if the hardware ID is changed on the user's PC, and even if the Allow Hardware Changes option is enabled, the registry item or file with registration information will become invalid and the application will require registration again;
- | Do not use this feature to lock registration information to a particular PC. To lock the license to a particular PC, use the [Hardware Lock](#) panel and generate the registration keys with user's hardware ID.

Storing data in the Windows registry

Use system registry - check this radio button to store registration data in the Windows registry. Attention: after the reinstallation of the operating system, the user will have to enter registration information once again.

Base place in registry - select the base branch of the registry which will hold the registration data. Possible values are:

- | HKEY_CURRENT_USER
- | HKEY_LOCAL_MACHINE

Relative path in registry - indicates the location where registration data will be stored. Use the following structure to fill out the field: `\SOFTWARE\YourCompany\`. The registration name will be stored at this address as the value of the "Name" parameter and the registration key will be stored at the same address as the value of "Key" parameter.

For example if you point out the base location in the registry as HKEY_CURRENT_USER and the relative path in the registry as `\SOFTWARE\YourCompany\`, the registration information will be stored at the following paths:

Registration name - HKEY_CURRENT_USER\SOFTWARE\YourCompany\Name

Registration key - HKEY_CURRENT_USER\SOFTWARE\YourCompany\Key

Storing data in an external file

Use file system - check this radio button to store registration data in an external file .

Base folder in file system - select the base path where file with registration data will be placed:

- | %DEFAULT FOLDER% - the folder where the protected module is placed (the same value if kept empty);
- | %SYSTEM FOLDER% - System32 (WinNt) or System (Win9X) subfolder of the Windows installation folder;
- | %WINDOWS FOLDER% - the Windows installation folder;
- | %My Documents FOLDER% - My Documents folder. Attention: For operating systems of the Windows NT family there is an individual "My Documents" folder for each user. Registration may not affect all users;
- | %My Pictures FOLDER% - My Pictures folder. It has the same warning as for My Documents folder;
- | %Program Files FOLDER% - Program Files folder;
- | %Program Files\Common FOLDER% - Program Files\Common folder;
- | %AllUsers\Documents FOLDER% - All Users\Documents folder;
- | %History FOLDER% - History folder. It has the same warning as for My Documents folder;
- | %Cookies FOLDER% - Cookies folder. It has the same warning as for My Documents folder;

- | %InternetCache FOLDER% - InternetCache folder. It has the same warning as for My Documents folder;
- | %ApplicationData FOLDER% - ApplicationData folder. It has the same warning as for My Documents folder.

Relative path in file system - a relative path where the file with registration data will be placed. If that path does not exist, the Enigma Loader will create the entire folder structure automatically when saving the key. Use the following structure to fill out the field: `\notepad\secret.dat`, where notepad is an additional folder relative base path, secret.dat - the name of the file where registration data will be stored, dat - the extension of the file. The format of that file is just an .ini file format.

Set File Attributes - allows you to set additional attributes to the license file. If this option is not selected, normal attributes will be applied to the license file.

Use of both techniques

Use both techniques - check this radio button to store registration data in the Windows registry and an external file simultaneously.

Hardware Lock

Select devices that will be used in hardware locking:

	Allow Changes
<input type="checkbox"/> System Volume Serial Number	0
<input type="checkbox"/> System Volume Name	0
<input type="checkbox"/> Computer Name	0
<input type="checkbox"/> CPU Type	0
<input type="checkbox"/> Motherboard	0
<input type="checkbox"/> Windows Serial Key	0
<input checked="" type="checkbox"/> Hard Disk Serial Number	0
<input checked="" type="checkbox"/> Windows User Name	1

This section provides possibilities for selecting the type of locking the registration key to system hardware or user data. The Enigma Protector has a set of features that provide possibilities to create registration keys locked to a specific computer. These keys will work on just one computer. You can find detailed information on using hardware-locked registration keys in the description of the [EP_RegHardwareID](#) function of the [Enigma API description section](#). See also the [Keys Generator](#) section.

Types of locking:

- | Volume Serial Drive - the serial number of the system partition of the hard drive;
- | System Volume Name - the name of the system partition of the hard drive;
- | Computer Name - the computer name (name of the currently active system user);
- | CPU type - the type of CPU;
- | Motherboard - information from the motherboard BIOS;
- | Windows Serial Key - the serial key of the installed Windows;
- | Windows User Name - the name of the currently active Windows user account.

Locking type	Is the same in different operating systems	Is the same after the hard disk drive is formatted or modified	Can be modified by the user	Can be the same on different computers
Volume Serial Drive	yes(*1)	no	yes (*4)	seldom
System Volume Name	yes(*1)	defined by user (*2)	yes	seldom
Computer Name	defined by user	defined by user (*3)	yes	seldom
CPU type (*5)	yes	yes	no	very often
Motherboard BIOS (*5)	yes	yes	no	seldom
Windows Serial Key (*7)	no	yes	no	seldom
Hard Disk Serial Number (*6)	yes	the same only after formatting	no	no
Windows User Name	defined by user	defined by user (*3)	yes	seldom

(*1) The condition is effective if operating systems are installed on the same partition of the hard disk drive and the system partition is not changed or formatted.

(*2) The condition is effective if the user enters the same partition name as it was before formatting.

(*3) Formatting or changing a system partition means reinstallation of the operating system. This condition can be accomplished if the user enters the same computer/user name that was in the previous operating system.

(*4) This change can be made only with the help of special software.

(*5) Administrator or Group privileges are needed on Windows NT (Windows 2000, Windows XP, Windows Vista).

(*6) Hard Disk Serial hardware lock failed to get the serial number if the operation system is installed on a RAID

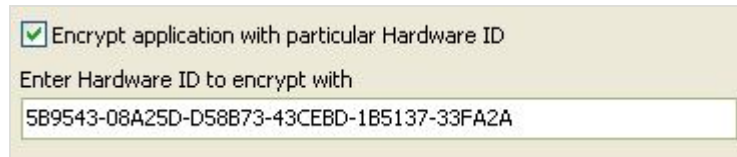
hard disk drive. Usually, the number of RAID disks is no more than 2-5% of all users.

(*7) Windows Serial Key hardware lock may return the same values on the corporate/company version of Windows. If your software mainly belongs to company users, it is recommended to use it together with Windows User Name hardware lock.

Advice: since there are a lot of hardware/software things to generate the hardware ID, it may be difficult to select the best configuration. Our recommendation is to initially use only Hard Disk Serial Number hardware lock. Hard Disk Serial Number remains the same after formatting and reinstalling the operation system, it can't be the same on multiple computers and has a small percentage of failures.

Allow Changes - specifies the number of selected hardware changes. This feature will keep the registration key valid, until the selected hardware is changed the selected number of times. Why can hardware ID be changed? This could happen due to execution permissions (for example, if a motherboard lock is enabled, and the program runs under the administrator for the first time, and without any administrator privileges for the second one, it causes hardware ID change) or due to hardware update (such as changes of the Hard Disk, motherboard, etc) or due to user changes (a computer name or system volume name). We recommend you to set the number of changes to at least 1 for each selected hardware.

Encrypt with Hardware ID



The image shows a software dialog box with a light beige background. At the top, there is a checked checkbox followed by the text "Encrypt application with particular Hardware ID". Below this, the text "Enter Hardware ID to encrypt with" is displayed. Underneath, a text input field contains the alphanumeric string "5B9543-08A25D-D58B73-43CEBD-1B5137-33FA2A".

This feature allows you to encrypt the file with the particular hardware ID. The purpose of this feature is to limit the file usage to only one particular PC, moreover, it will be really impossible to run/unpack/crack the file if the hardware ID (which the file is encrypted with) is unknown. It will be possible to run the protected file ONLY on the PC that has the same hardware ID as the one it was encrypted with.

To apply this feature, it is very helpful to use some kind of an activator program. Run on the user's PC, this program will generate and show the Hardware ID. Then this ID should be sent to the developer who protects personal, hardware encrypted copy of your application.

If there is any attempt to run the file on the PC with a different Hardware ID, the execution will depend on the properties set in the CHECK-UP - Control Sum panel, i.e. if the control sum check-up is enabled, the Control Sum message will be shown. If it is disabled, the application will be terminated silently.

Registration Dialog



The screenshot shows a configuration window for the Registration Dialog. It has a light beige background and a title bar. The window contains the following elements:

- A checked checkbox labeled "Use custom registration dialog".
- A section titled "Show dialog properties" containing two checkboxes: "Show if the trial has expired" (unchecked) and "Show if unregistered" (checked).
- A section titled "Dialog Messages" containing two checkboxes: "Show message if key is valid" (unchecked) and "Show message if key is invalid" (unchecked). To the right of each checkbox is a "Design message" button.
- A "Design" button at the bottom center of the window.

The current feature allows you to add a custom registration dialog to the executable even without modifying or recompiling the main module. Note that this Registration Dialog can also be called by means of the Enigma API function [EP_RegShowDialog](#).

Use custom registration dialog - enables the registration dialog to be shown.

Show if the trial has expired - show the registration dialog only if the trial period of the protected module has expired.

Show if unregistered - show the registration dialog if the application is unregistered.

Show message if key is valid - show a message if the user has succeeded with the registration and entered the correct registration information. Use the "Design Message" button to specify message texts.

Show message if key is invalid - show a message if the user has not succeeded with the registration and entered incorrect registration information. Use the "Design Message" button to specify message texts.

Design - press this button to design a custom registration dialog. Follow the link to learn more about [Registration Dialog Designer](#).

Align Property

Determines how the control aligns within its container (parent control).

Description

Use Align to align a control to the top, bottom, left, or right of a form or panel and have it remain there even if the size of the form, panel, or component that contains the control changes. When the parent is resized, an aligned control also resizes so that it continues to span the top, bottom, left, or right edge of the parent.

For example, to use a panel component with various controls on it as a tool palette, change the panel's Align value to `alLeft`. The value of `alLeft` for the Align property of the panel guarantees that the tool palette remains on the left side of the form and always equals the client height of the form.

The default value of Align is `alNone`, which means a control remains where it is positioned on a form or panel.

Tip: If Align is set to `alClient`, the control fills the entire client area so that it is impossible to select the parent form by clicking on it. In this case, select the parent by selecting the control on the form and pressing Esc, or by using the Object Inspector.

Any number of child components within a single parent can have the same Align value, in which case they stack up along the edge of the parent. To adjust the order in which the controls stack up, drag the controls into their desired positions.

Note: To cause a control to maintain a specified relationship with an edge of its parent, but not necessarily lie along one edge of the parent, use the Anchors property instead.

Value	Meaning
<code>alNone</code>	The control remains where it was placed. This is the default value
<code>alTop</code>	The control moves to the top of its parent and resizes to fill the width of its parent. The height of the control is not affected
<code>alBottom</code>	The control moves to the bottom of its parent and resizes to fill the width of its parent. The height of the control is not affected
<code>alLeft</code>	The control moves to the left side of its parent and resizes to fill the height of its parent. The width of the control is not affected
<code>alRight</code>	The control moves to the right side of its parent and resizes to fill the height of its parent. The width of the control is not affected
<code>alClient</code>	The control resizes to fill the client area of its parent. If another control already occupies part of the client area, the control resizes to fit within the remaining client area
<code>alCustom</code>	The control's positioning is determined by calls to its parent's <code>CustomAlignInsertBefore</code> and <code>CustomAlignPosition</code> methods

Alignment Property

Controls the horizontal placement of the text within the label.

Description

Set Alignment to specify how the text of the label is justified within the ClientRect of the label control.

The effect of the Alignment property is more obvious if the WordWrap property is true and the label includes more than one line of text.

AlphaBlend Property

Specifies whether the form is translucent.

Description

Set `AlphaBlend` to specify that the form represents a layered window that allows a translucent color. The `AlphaBlendValue` property specifies the degree of translucency.

AlphaBlendValue Property

Specifies the degree of translucency on a translucent form.

Description

Set AlphaBlendValue to a value between 0 and 255 to indicate the degree of translucency when the AlphaBlend property is true. A value of 0 indicates a completely transparent window. A value of 255 indicates complete opacity.

Note: AlphaBlendValue only has an effect when the AlphaBlend property is true.

Anchors Property

Specifies how the control is anchored to its parent.

Description

Use Anchors to ensure that a control maintains its current position relative to an edge of its parent, even if the parent is resized. When its parent is resized, the control holds its position relative to the edges to which it is anchored.

If a control is anchored to opposite edges of its parent, the control stretches when its parent is resized. For example, if a control has its Anchors property set to [akLeft, akRight], the control stretches when the width of its parent changes.

Anchors is enforced only when the parent is resized. Thus, for example, if a control is anchored to opposite edges of a form at design time and the form is created in a maximized state, the control is not stretched because the form is not resized after the control is created.

Note: If a control should maintain contact with three edges of its parent (hugging one side of the parent and stretching the length of that side), use the Align property instead. Unlike Anchors, Align allows controls to adjust to changes in the size of other aligned sibling controls as well as changes to the parent's size.

Value	Meaning
akTop	The control's position is fixed with respect to the top edge of its parent
akLeft	The control's position is fixed with respect to the left edge of its parent
akRight	The control's position is fixed with respect to the right edge of its parent
akBottom	The control's position is fixed with respect to the bottom edge of its parent

AutoSize Property

Specifies whether the control sizes itself automatically to accommodate its contents.

Description

Use `AutoSize` to specify whether the control sizes itself automatically. When `AutoSize` is true, the control resizes automatically when its contents change.

BevelEdges Property

Specifies which edges of the control are beveled.

Description

The BevelInner, BevelOuter, and BevelKind properties determine the appearance of the specified edges.

Value	Meaning
beLeft	The left edge is beveled.
beTop	The top edge is beveled.
beRight	The right edge is beveled.
beBottom	The bottom edge is beveled.

BevelInner Property

Specifies the cut of the inner bevel.

Description

Use `BevelInner` to specify whether the inner bevel has a raised, lowered, or flat look.

The inner bevel appears immediately inside the outer bevel. If there is no outer bevel (`BevelOuter` is `bvNone`), the inner bevel appears immediately inside the border.

Value	Meaning
<code>bvNone</code>	The bevel does not exist.
<code>bvLowered</code>	The bevel appears lowered.
<code>bvRaised</code>	The bevel appears raised.
<code>bvSpace</code>	The bevel appears as a space if its kind is not <code>bkTile</code> . Otherwise, the bevel appears raised.

BevelKind Property

Specifies the control's bevel style.

Description

Use `BevelKind` to modify the appearance of a bevel. `BevelKind` influences how sharply the bevel stands out.

`BevelKind`, in combination with `BevelWidth` and the cut of the bevel specified by `BevelInner` or `BevelOuter`, can create a variety of effects. Experiment with various combinations to get the look you want.

Value	Meaning
<code>bkNone</code>	No bevel is added.
<code>bkTile</code>	The bevel appears sharply defined.
<code>bkSoft</code>	The bevel uses softer contrasts than <code>bkTile</code> .
<code>bkFlat</code>	The bevel has a broad, flat appearance.

BevelOuter Property

Specifies the cut of the outer bevel.

Description

Use `BevelInner` to specify whether the outer bevel has a raised, lowered, or flat look.

The outer bevel appears immediately inside the border and outside the inner bevel.

Value	Meaning
<code>bvNone</code>	The bevel does not exist.
<code>bvLowered</code>	The bevel appears lowered.
<code>bvRaised</code>	The bevel appears raised.
<code>bvSpace</code>	The bevel appears as a space if its kind is not <code>bkTile</code> . Otherwise, the bevel appears raised.

BevelWidth Property

Determines the width, in pixels, of both the inner and outer bevels of a panel.

Description

Use `BevelWidth` to specify how wide the inner or outer bevel should be. Do not confuse `BevelWidth`, which is the width of the bevels, with `BorderWidth`, which is the space between the bevels.

If both the `BevelInner` and `BevelOuter` properties are `bvNone`, `BevelWidth` has no effect. To remove both bevels, set the `BevelInner` and `BevelOuter` properties to `bvNone`, rather than setting the `BevelWidth` to 0, as this involves less overhead when painting.

BiDiMode Property

Specifies the bi-directional mode for the control.

Description

The bi-directional mode controls the reading order for the text, the placement of the vertical scroll bar, and whether the alignment is changed.

Value	Meaning
bdLeftToRight	Reading order is left to right. Alignment is not changed. The vertical scroll bar appears on the right edge of the control.
bdRightToLeft	Reading order is right to left. Alignment is changed. The vertical scroll bar appears on the left edge of the control.
bdRightToLeftNoAlign	Reading order is right to left. Alignment is not changed. The vertical scroll bar appears on the left edge of the control.
bdRightToLeftReadingOnly	Reading order is right to left. Alignment and scroll bar are not changed.

BorderIcons Property

Specifies which icons appear on the title bar of the form.

Description

May contain the following values:

Value	Meaning
biSystemMenu	The form has a Control menu (also known as a System menu).
biMinimize	The form has a Minimize button
biMaximize	The form has a Maximize button
biHelp	If BorderStyle is bsDialog or biMinimize and biMaximize are excluded, a question mark appears in the form's title bar and when clicked, the cursor changes to crHelp; otherwise, no question mark appears.

BorderStyle Property

Determines whether the edit control has a single line border around the client area.

Description

Use `BorderStyle` to affect the sharpness with which the client area of the edit control stands out. `BorderStyle` can have a value of either `bsSingle` or `bsNone`. If `BorderStyle` is `bsSingle`, the edit control has a single-line border around the client area. If `BorderStyle` is `bsNone`, there will be no border.

Value	Meaning
<code>bsNone</code>	No visible border.
<code>bsSingle</code>	Single-line border.

BorderWidth Property

Specifies the distance, in pixels, between the outer and inner bevels.

Description

Use `BorderWidth` to specify how wide the border around the panel should be. A value of 0 (zero) means no border should appear.

Brush Property

Specifies the color and pattern used for filling the shape control.

Cancel Property

Determines whether the button's OnClick event handler executes when the Escape key is pressed.

Description

If Cancel is true, the button's OnClick event handler executes when the user presses Esc. Although an application can have more than one Cancel button, the form calls the OnClick event handler only for the first visible button in the tab order.

Caption Property

Specifies a text string that identifies the control to the user.

Description

Use Caption to specify the text string that labels the control.

To underline a character in a Caption that labels a component, include an ampersand (&) before the character. This type of character is called an accelerator character. The user can then select the component by pressing Alt while typing the underlined character. To display an ampersand character in the caption, use two ampersands (&&).

Note: Controls that display text use either the Caption property or the Text property to specify the text value. Which property is used depends on the type of control. In general, Caption is used for text that appears as a window title or label, while Text is used for text that appears as the content of a control.

Center Property

Indicates whether the image is centered in the image control.

Description

When the image does not fit perfectly within the image control, use Center to specify how the image is positioned. When Center is true, the image is centered in the control. When Center is false, the upper left corner of the image is positioned at the upper left corner of the control.

CharCase Property

Determines the case of the text within the edit control.

Description

Value	Meaning
ecLowerCase	The text is converted to lowercase.
ecNormal	The text appears in mixed case. It is not forced into any case.
ecUpperCase	The text is converted to uppercase.

ClickAction Property

Determines the OnClick event handler.

Description

ClickAction may have following values.

Value	Action
caRegister	Perform register. This way the application reads name and key value from edit controls and tries to register.
caContinue	Continue execution
caExit	Exit application
caNone	Nothing to do

ClientHeight Property

Specifies the height (in pixels) of the form's client area.

Description

Use `ClientHeight` to determine the height (in pixels) of the form's client area. The client area is the usable area inside the form's border, excluding the title bar, scroll bars, and so on.

Set `ClientHeight` to change the height of the form's window based on the desired client area. To change the height of the form's window based on the total size of the window (including the border, menu, status bar and so on), use the `Height` property instead.

ClientWidth Property

Specifies the width (in pixels) of the form's client area.

Description

Use ClientWidth to determine the width (in pixels) of the form's client area. The client area is the usable area inside the form's border. Set ClientWidth to change the width of the form's window based on the desired client area. To change the width of the form's window based on the total size of the window (including the border, status bar and so on), use the Width property instead.

CloseAction Property

Determines the OnClose event handler.

Description

CloseAction may have following values.

Value	Action
caRegister	Perform register. This way the application reads name and key value from edit controls and tries to register.
caContinue	Continue execution
caExit	Exit application
caNone	Nothing to do

Color Property

Specifies the background color of the control.

Description

Use Color to read or change the background color of the control.

Constraints Property

Specifies the size constraints for the control.

Description

Use Constraints to specify the minimum and maximum width and height of the control. When Constraints contains maximum or minimum values, the control cannot be resized to violate those constraints.

Warning: Do not set up constraints that conflict with the value of the Align or Anchors property. When these properties conflict, the response of the control to resize attempts is not well-defined.

Content Property

Determines the content of the edit control. This information is used when the registration information is verifying and the current hardware id should be shown.

Description

Content may have following values.

Value	Action
coRegistratioName	The edit will contain a registration name. Users will have to enter registration name into this control.
coRegistratioKey	The edit will contain a registration key. Users will have to enter registration key into this control.
coHardwareID	This edit will be filled out with the Hardware ID.
coCustom	The edit contains custom text.

Ctl3D Property





















Determines whether a control has a three-dimensional (3-D) or two-dimensional look.

Cursor Property

Specifies the image used to represent the mouse pointer when it passes into the region covered by the control.

Description

Change the value of Cursor to provide feedback to the user when the mouse pointer enters the control.

Cursor	Image
crDefault	Whatever cursor is the default for the window class (usually crArrow)
crNone	
crArrow	
crCross	
crIBeam	
crSizeNESW	
crSizeNS	
crSizeNWSE	
crSizeWE	
crUpArrow	
crHourGlass	
crDrag	
crNoDrop	
crHSplit	
crVSplit	
crMultiDrag	
crSQLWait	
crNo	
crAppStart	
crHelp	
crHandPoint	

crSize	
crSizeAll	

Default Property

Determines whether the button's OnClick event handler executes when the Enter key is pressed.

Description

If Default is true, the button's OnClick event handler executes when the user presses Enter.

Although an application can have more than one Default button, the form calls the OnClick event handler only for the first visible button in the tab order. Moreover, any button that has focus becomes the Default button temporarily; hence, if the user selects another button before pressing Enter, the selected button's OnClick event handler executes instead.

Enabled Property

Controls whether the control responds to mouse and keyboard.

Description

Use Enabled to change the availability of the control to the user. To disable a control, set Enabled to false. Disabled controls appear dimmed. If Enabled is false, the control ignores mouse and keyboard.

Flat Property

Determines whether the button has a 3D border that provides a raised or lowered look.

Description

Set Flat to true to remove the raised border when the button is unselected and the lowered border when the button is clicked or selected. When Flat is true, use separate bitmaps for the different button states to provide visual feedback to the user about the button state.

Font Property

Controls the attributes of text written on or in the control.

FontHovered Property

Controls the attributes of text of the TLink when the mouse pointer is moving over the control.

FormStyle Property

Determines the form's style.

Description

FormStyle consists of the following values.

Value	Meaning
fsNormal	The form is neither an MDI parent window nor an MDI child window.
fsMDIChild	The form is an MDI child window.
fsMDIForm	The form is an MDI parent window.
fsStayOnTop	This form remains on top of the desktop and of other forms in the project, except any others that also have FormStyle set to fsStayOnTop. If one fsStayOnTop form launches another, neither form will consistently remain on top.

Glyph Property

Specifies the bitmap that appears on the speed button.

Description

Set Glyph to a bitmap object that contains the image that should appear on the face of the button. Bring up the Open dialog box from the Object Inspector to choose a bitmap file (with a .BMP extension).

Glyph can provide up to four images within a single bitmap. All images must be the same size and next to each other in a horizontal row. TSpeedButton displays one of these images depending on the state of the button.

Image position	Button state	Description
First	Up	This image appears when the button is unselected. If no other images exist in the bitmap, this image is used for all states.
Second	Disabled	This image usually appears dimmed to indicate that the button can't be selected.
Third	Clicked	This image appears when the button is clicked. If GroupIndex is 0, the Up image reappears when the user releases the mouse button.
Fourth	Down	This image appears when the button stays down indicating that it remains selected.

If only one image is present, TSpeedButton attempts to represent the other states by altering the image slightly for each state, although the Down state is always the same as the Up state.

If the bitmap contains multiple images, specify the number of images in the bitmap with the NumGlyphs property.

Note: The lower left pixel of the bitmap is reserved for the "transparent" color. Any pixel in the bitmap that matches the lower left pixel will be transparent.

Height Property

Specifies the vertical size of the control in pixels.

Hint Property

Contains the text string that can appear when the user moves the mouse over the control.

Description

Use the Hint property to provide a string of help text either as a Help Hint, or as help text on a particular location such as a status bar.

A Help Hint is a box containing help text that appears for a control when the user moves the mouse pointer over the control and pauses momentarily. To set up Help Hints:

- Specify the Hint property of each control for which a Help Hint should appear
- Set the ShowHint property of each appropriate control to true and set the ShowHint property of the form to true.

Note: If the application's ShowHint property is false, the Help Hint does not appear.

HorzScrollBar Property

Represents the horizontal scroll bar for the scrolling windowed control.

Layout Property

Specifies the vertical placement of the text within the label.

Description

Set `Layout` to specify how the text of the label is placed within the `ClientRect` of the label control. `Layout` is the vertical analog to the `Alignment` property.

Left Property

Specifies the horizontal coordinate of the left edge of a component relative to its parent.

Lines Property

Contains the individual lines of text in the memo control.

Description

Use Lines to manipulate text in an memo control on a line-by-line basis.

Margin Property

Specifies the number of pixels between the edge of the button and the image or caption drawn on its surface.

Description

Use Margin to specify the indentation of the image specified by the Glyph property or the text specified by the Caption property. The edges that Margin separates depends on the Layout property. If Layout is blGlyphLeft, the margin appears between the left edge of the image or caption and the left edge of the button. If Layout is blGlyphRight, the margin separates the right edges. If Layout is blGlyphTop, the margin separates the top edges, and if Layout is blGlyphBottom, the margin separates the bottom edges.

If Margin is -1, the image or text is centered on the button.

MaxLength Property

Specifies the maximum number of characters the user can enter into the edit control.

Description

Use `MaxLength` to limit the number of characters that can be entered into the edit control. A value of `-1` indicates that there is no application-defined limit on the length.

Use `MaxLength` to limit the length of the text in an edit control if that text will be copied into a fixed-length buffer.

Note: Setting `MaxLength` to a value less than the number of characters currently in the edit control causes the edit control to truncate its text to `MaxLength` characters.

Name Property

Specifies the name of the control.

Description

Use the Name property to assign a new name to the control or to find out what the name of the control is. Property is informative, it does not affect on workability.

NumGlyphs Property

Specifies the number of images included in the Glyph property.

Description

Set NumGlyphs to the number of images provided by the bitmap assigned to the Glyph property. All images must be the same size and next to each other in a row. The Glyph property can provide up to four images.

PasswordChar Property

Indicates the character, if any, to display in place of the actual characters typed in the control.

Description

Use the PasswordChar property to create an edit control that displays a special character in place of any entered text. If PasswordChar is set to the null character (ANSI character zero), the edit control displays its text normally. If PasswordChar is any other character, the edit control displays PasswordChar in place of each character typed. PasswordChar affects the appearance of the edit control only. The value of the Text property reflects the actual characters that are typed.

Pen Property

Specifies the pen used to outline the shape control.

Picture Property

Specifies the image that appears on the image control.

Position Property

Represents the size and placement of the form.

Proportional Property

Indicates whether the image should be changed, without distortion, so that it fits the bounds of the image control.

Description

Set Proportional to true to ensure that the image can be fully displayed in the image control without any distortion such as occurs with the Stretch property. When Proportional is true, images that are too large to fit in the image control are scaled down (while maintaining the same aspect ratio) until they fit in the image control. Images that are too small are displayed normally. That is, Proportional can reduce the magnification of the image, but does not increase it.

When the image control resizes, the image resizes also.

To resize the image so that it fits exactly in the image control, even if that causes distortion, use the Stretch property instead.

To resize the control to the image rather than resizing the image to the control, use the AutoSize property instead.

The default value for Proportional is false.

Note: Proportional has no effect if the Picture property contains an icon.

ReadOnly Property

Determines whether the user can change the text of the edit control.

Description

To restrict the edit control to display only, set the ReadOnly property to true. Set ReadOnly to false to allow the contents of the edit control to be edited.

ScrollBars Property

Determines whether the control has scroll bars.

Description

ScrollBars can take one of the following values:

Value	Meaning
ssNone	The control has no scroll bars.
ssHorizontal	The control has a single scroll bar on the bottom edge.
ssVertical	The control has a single scroll bar on the right edge.
ssBoth	Horizontal scrollbar that appears as-needed.

Shape Property

Determines the shape of the bevel.

Description

Set Shape to specify whether the bevel appears as a line, box, frame, or space. For shapes that can appear either raised or lowered, the Style property indicates which effect is used.

The default value for Shape is bsBox.

Value	Meaning
bsBox	The entire client area appears raised or lowered.
bsFrame	The client area is outlined by a raised or lowered frame.
bsTopLine	The bevel displays a line at the top of the client area.
bsBottomLine	The bevel displays a line at the bottom of the client area.
bsLeftLine	The bevel displays a line at the left side of the client area.
bsRightLine	The bevel displays a line at the right side of the client area.
bsSpacer	The bevel is an empty space.

ShowCopyButton Property

Allows to show a Copy to Clipboard button near with the edit control and associated with it.

Description

This property could be, for example, used to copy shown Hardware ID from control to clipboard.

ShowHint Property

Determines whether the control displays a Help Hint when the mouse pointer rests momentarily on the control.

Description

The Help Hint is the value of the Hint property, which is displayed in a box just beneath the control. Use ShowHint to determine whether a Help Hint appears for the control.

To enable Help Hint for a particular control, the application ShowHint property must be true.

ShowPasteButton Property

Allows to show a Paste from Clipboard button near with the edit control and associated with it.

Description

This property could be, for example, used to paste registration name or key to the control from clipboard.

Spacing Property

Determines where the image and text appear on a speed button.

Description

Set Spacing to the number of pixels that should appear between the image specified in the Glyph property and the text specified in the Caption property.

If Spacing is a positive number, its value is the number of pixels between the image and text. If Spacing is 0, the image and text appear flush with each other. If Spacing is -1, the text appears centered between the image and the button edge.

Stretch Property

Indicates whether the image should be changed so that it exactly fits the bounds of the image control.

Description

Set `Stretch` to `true` to cause the image to assume the size and shape of the image control. When the image control resizes, the image resizes also. `Stretch` resizes the height and width of the image independently. Thus, unlike a simple change in magnification, `Stretch` can distort the image if the image control is not the same shape as the image.

To resize the control to the image rather than resizing the image to the control, use the `AutoSize` property instead.

Style Property

Determines whether the bevel appears raised or lowered.

Description

Set `Style` to indicate whether the bevel should create a raised or a lowered effect. When the `Shape` property is `bsBox`, the entire client area appears raised or lowered. For all other values of `Shape`, the bevel displays a raised or lowered line along the edge or edges of the client area. The default value of `Style` is `bsLowered`.

Value	Meaning
<code>bsLowered</code>	The bevel is lowered.
<code>bsRaised</code>	The bevel is raised.

TabOrder Property

Indicates the position of the control in its parent's tab order.

Description

TabOrder is the order in which child windows are visited when the user presses the Tab key. The control with the TabOrder value of 0 is the control that has the focus when the form first appears.

Initially, the tab order is always the order in which the controls were added to the form. The first control added to the form has a TabOrder value of 0, the second is 1, the third is 2, and so on. Change this by changing the TabOrder property.

Each control has a unique tab-order value within its parent. If you change the TabOrder property value of one control to be the same as the value of a second control, the TabOrder value for all the other controls changes. For example, suppose a control is sixth in the tab order. If you change the control's TabOrder property value to 3 (making the control fourth in the tab order), the control that was originally fourth in the tab order now becomes fifth, and the control that was fifth becomes sixth.

Assigning TabOrder a value greater than the number of controls contained in the parent control moves the control to the end of the tab order. The control does not take on the assigned value of TabOrder, but instead is given the number that assures the control is the last in the tab order.

Note: TabOrder is meaningful only if the TabStop property is true and if the control has a parent. (The TabOrder property of a form is not used unless the form is the child of another form.) A control with a TabOrder of -1 has no parent, and therefore cannot be reached by pressing the Tab key. To remove a parented control from the Tab order, set its TabStop property to false.

TabStop Property

Determines if the user can tab to a control.

Description

Use the TabStop to allow or disallow access to the control using the Tab key.

If TabStop is true, the control is in the tab order. If TabStop is false, the control is not in the tab order and the user can't press the Tab key to move to the control.

Note: TabStop is not meaningful for a form unless the form assigns another form to be its parent.

Top Property

Specifies the Y coordinate of the top left corner of a control, relative to its parent or containing control in pixels.

Transparent Property

Specifies whether the background of the control obscures objects below the control object.

Description

Set Transparent to true to allow objects behind the control object to show through the background of the control.

Set Transparent to false to make the background of the control opaque.

TransparentColor Property

Specifies whether a color on the form appears transparent.

Description

Use `TransparentColor` to indicate that one of the colors on the form should be treated as transparent, allowing windows behind the form to completely show through. The `TransparentColorValue` property indicates the color that appears completely transparent.

Note: To make the entire form transparent, or to make it translucent rather than transparent, use the `AlphaBlend` and `AlphaBlendValue` properties.

TransparentColorValue Property

Indicates the color on the form that appears transparent when TransparentColor is true.

Description

Use TransparentColorValue to indicate the color that appears transparent when the TransparentColor property is true.

Url Property

Specifies the Url that will be opened on OnClick action.

Description

Url property can open not only the browser window but also a default email client and some other files. For example:

url `http://enigmaprotector.com/` opens a browser window with the current address,

url `mailto:support@enigmaprotector.com` opens a default email client with the offer to create email for `support@enigmaprotector.com`

url `License.txt` opens a Windows notepad with the `License.txt` file.

VertScrollBar Property

Represents the vertical scroll bar for the scrolling windowed control.

Visible Property

Determines whether the component appears onscreen.

WantReturns Property

Determines whether the user can insert return characters into the text.

Description

Set `WantReturns` to true to allow users to enter return characters into the text. Set `WantReturns` to false to allow the form to handle return characters instead.

For example, in a form with a default button (such as an OK button) and a memo control, if `WantReturns` is false, pressing Enter chooses the default button. If `WantReturns` is true, pressing Enter inserts a return character in the text.

Note: If `WantReturns` is false, users can still enter return characters into the text by pressing Ctrl+Enter.

Width Property

Specifies the horizontal size of the control or form in pixels.

WindowState Property

Represents how the form appears on the screen.

Description

WindowState describes the state of a form window. The following table lists the possible values:

Value	Meaning
wsNormal	The form is in its normal state (that is, neither minimized nor maximized).
wsMinimized	The form is minimized.
wsMaximized	The form is maximized.

WordWrap Property

Specifies whether the button text wraps to fit the width of the control.

Description

Set `WordWrap` to `true` to allow the label to display multiple line of text. When `WordWrap` is `true`, text that is too wide for the control wraps at the right margin.

Set `WordWrap` to `false` to limit the label to a single line. When `WordWrap` is `false`, text that is too wide for the label appears truncated.

TBevel



Use TBevel to create beveled boxes, frames, or lines. The bevel can be raised or lowered.

Properties

- | [Align](#)
- | [Anchors](#)
- | [Constraints](#)
- | [Cursor](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [Name](#)
- | [Shape](#)
- | [ShowHint](#)
- | [Style](#)
- | [Top](#)
- | [Visible](#)
- | [Width](#)

TButton

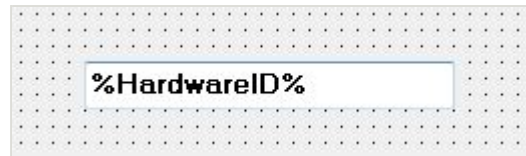


Use TButton to put a standard push button on the form. TButton may process such user actions like register, close application and continue execution.

Properties

- | [Anchors](#)
- | [BiDiMode](#)
- | [Cancel](#)
- | [Caption](#)
- | [ClickAction](#)
- | [Constraints](#)
- | [Cursor](#)
- | [Default](#)
- | [Enabled](#)
- | [Font](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [Name](#)
- | [ShowHint](#)
- | [TabOrder](#)
- | [TabStop](#)
- | [Top](#)
- | [Visible](#)
- | [Width](#)
- | [WordWrap](#)

TEdit

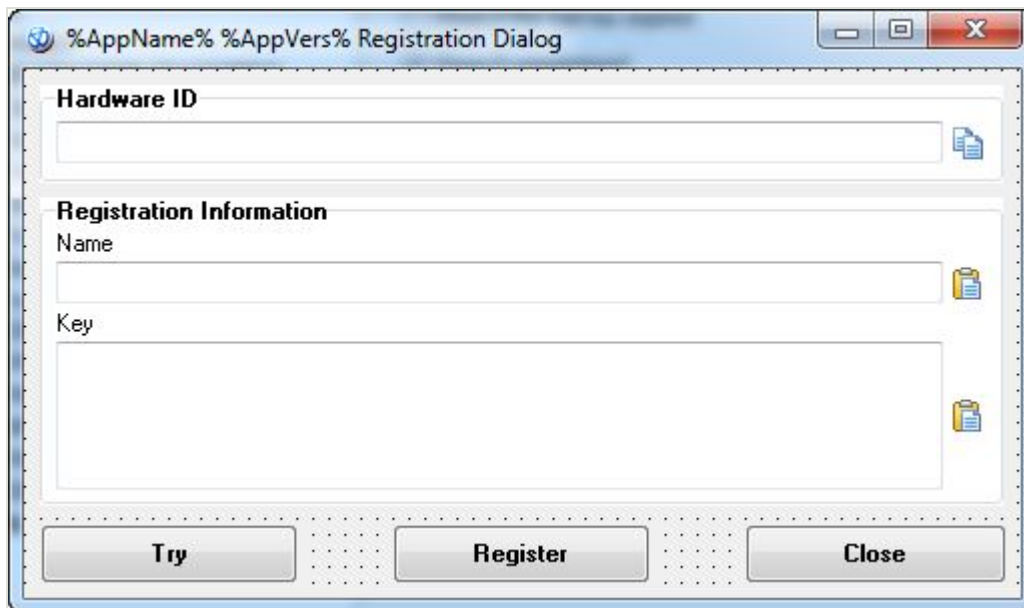


Use a TEdit object to put standard Windows edit control on the form. Edit controls are used to retrieve text that users type. The edit controls can also display the text to the user. TEdit may contain Hardware ID string or custom string, allows to enter registration name and key.

Properties

- | [Align](#)
- | [Anchors](#)
- | [BevelEdges](#)
- | [BevelInner](#)
- | [BevelKind](#)
- | [BevelOuter](#)
- | [BevelWidth](#)
- | [BiDiMode](#)
- | [BorderStyle](#)
- | [CharCase](#)
- | [Color](#)
- | [Constraints](#)
- | [Content](#)
- | [Ctl3D](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Font](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [MaxLength](#)
- | [Name](#)
- | [PasswordChar](#)
- | [ReadOnly](#)
- | [ShowCopyButton](#)
- | [ShowHint](#)
- | [ShowPasteButton](#)
- | [TabOrder](#)
- | [TabStop](#)
- | [Top](#)
- | [Visible](#)
- | [Width](#)

TForm

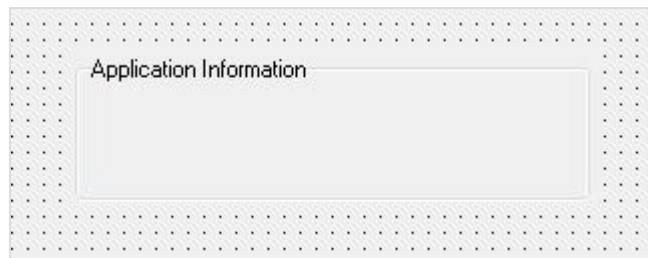


TForm represents the main registration window.

Properties

- | AlphaBlend
- | AlphaBlendValue
- | BiDiMode
- | BorderIcons
- | BorderStyle
- | BorderWidth
- | Caption
- | ClickAction
- | ClientHeight
- | ClientWidth
- | CloseAction
- | Color
- | Constraints
- | Ctl3D
- | Cursor
- | Enabled
- | Font
- | FormStyle
- | Height
- | Hint
- | HorzScrollBar
- | Left
- | Name
- | Position
- | ShowHint
- | Top
- | TransparentColor
- | TransparentColorValue
- | VertScrollBar
- | Width
- | WindowState

TGroupBox



The TGroupBox component represents standard Windows group box, used to group related controls in the form.

Properties

- | [Align](#)
- | [Anchors](#)
- | [BiDiMode](#)
- | [Caption](#)
- | [Color](#)
- | [Constraints](#)
- | [Ctl3D](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Font](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [Name](#)
- | [ShowHint](#)
- | [TabOrder](#)
- | [TabStop](#)
- | [Top](#)
- | [Visible](#)
- | [Width](#)

TImage



Use TImage to display a graphical image on the form.

Properties

- | [Align](#)
- | [Anchors](#)
- | [AutoSize](#)
- | [Center](#)
- | [Constraints](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [Name](#)
- | [Picture](#)
- | [Proportional](#)
- | [ShowHint](#)
- | [Stretch](#)
- | [Top](#)
- | [Transparent](#)
- | [Visible](#)
- | [Width](#)

TLabel



Use TLabel to add text that the user can't edit in the form.

Properties

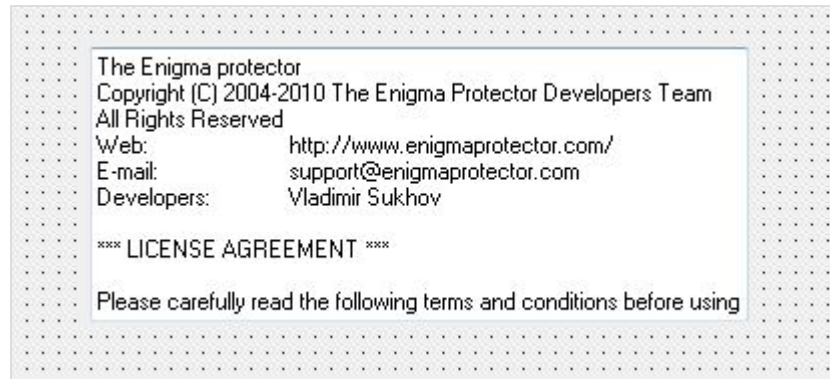
- | [Align](#)
- | [Alignment](#)
- | [Anchors](#)
- | [AutoSize](#)
- | [BiDiMode](#)
- | [Caption](#)
- | [Color](#)
- | [Constraints](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Font](#)
- | [Height](#)
- | [Hint](#)
- | [Layout](#)
- | [Left](#)
- | [Name](#)
- | [ShowHint](#)
- | [Top](#)
- | [Transparent](#)
- | [Visible](#)
- | [Width](#)
- | [WordWrap](#)

[Click Here to Go to the Online Order Page](#)

Use TLink for a hyperlink to the form that will be able to open an url in the browser, open default email client, open some external file.

Properties

- | [Align](#)
- | [Alignment](#)
- | [Anchors](#)
- | [AutoSize](#)
- | [BiDiMode](#)
- | [Caption](#)
- | [Color](#)
- | [Constraints](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Font](#)
- | [FontHovered](#)
- | [Height](#)
- | [Hint](#)
- | [Layout](#)
- | [Left](#)
- | [Name](#)
- | [ShowHint](#)
- | [Top](#)
- | [Transparent](#)
- | [Url](#)
- | [Visible](#)
- | [Width](#)
- | [WordWrap](#)



Use TMemo to put a standard Windows multiline edit control on the form. Multiline edit boxes allow the user to enter more than one line of text. They are appropriate for representing lengthy information. Memo may contain Hardware ID string or custom string, allows to enter registration name and key.

Properties

- | [Align](#)
- | [Alignment](#)
- | [Anchors](#)
- | [BevelEdges](#)
- | [BevelInner](#)
- | [BevelKind](#)
- | [BevelOuter](#)
- | [BiDiMode](#)
- | [BorderStyle](#)
- | [Color](#)
- | [Constraints](#)
- | [Content](#)
- | [Ctl3D](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Font](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [Lines](#)
- | [MaxLength](#)
- | [Name](#)
- | [ReadOnly](#)
- | [ScrollBars](#)
- | [ShowCopyButton](#)
- | [ShowHint](#)
- | [ShowPasteButton](#)
- | [TabOrder](#)
- | [TabStop](#)
- | [Top](#)
- | [Visible](#)
- | [WantReturns](#)
- | [Width](#)
- | [WordWrap](#)

TPanel

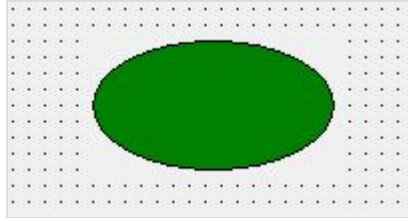


Use TPanel to put an empty panel on the form.

Properties

- | [Align](#)
- | [Alignment](#)
- | [Anchors](#)
- | [BevelEdges](#)
- | [BevelInner](#)
- | [BevelKind](#)
- | [BevelOuter](#)
- | [BevelWidth](#)
- | [BiDiMode](#)
- | [BorderStyle](#)
- | [BorderWidth](#)
- | [Caption](#)
- | [Color](#)
- | [Constraints](#)
- | [Ctl3D](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Font](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [Name](#)
- | [ShowHint](#)
- | [TabOrder](#)
- | [TabStop](#)
- | [Top](#)
- | [Visible](#)
- | [Width](#)

TShape



Add a TShape object to the form to draw a simple geometric shape on the form.

Properties

- | [Align](#)
- | [Anchors](#)
- | [Brush](#)
- | [Constraints](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Height](#)
- | [Hint](#)
- | [Left](#)
- | [Name](#)
- | [Pen](#)
- | [Shape](#)
- | [ShowHint](#)
- | [Top](#)
- | [Visible](#)
- | [Width](#)

TSpeedButton



TSpeedButton is almost same [TButton](#) control but allows to add an image onto the button.

Properties

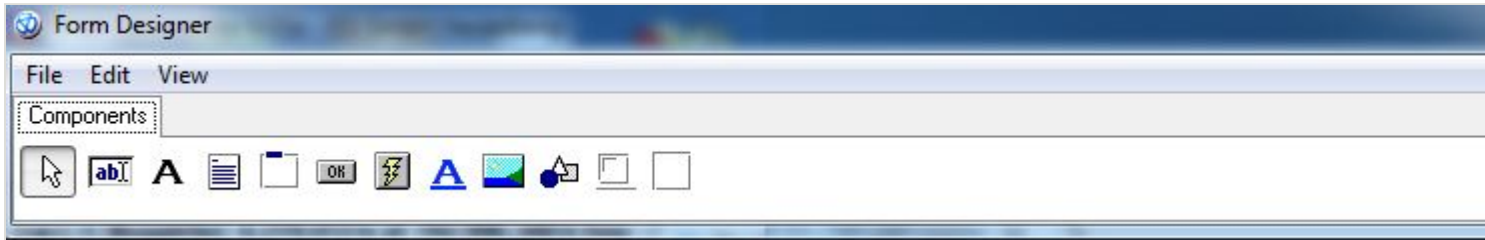
- | [Anchors](#)
- | [BiDiMode](#)
- | [Caption](#)
- | [ClickAction](#)
- | [Constraints](#)
- | [Cursor](#)
- | [Enabled](#)
- | [Flat](#)
- | [Font](#)
- | [Glyph](#)
- | [Height](#)
- | [Hint](#)
- | [Layout](#)
- | [Left](#)
- | [Margin](#)
- | [Name](#)
- | [NumGlyphs](#)
- | [ShowHint](#)
- | [Spacing](#)
- | [Top](#)
- | [Visible](#)
- | [Width](#)

Registration Dialog Designer












The Registration Dialog Designer allows you to design a custom registration dialog, to apply your own styles and configuration. Here you can put your own controls in the form and define registration events. The registration dialog forms:

Components Palette

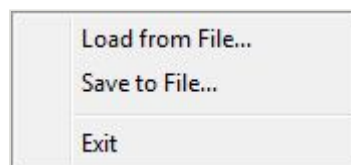
Contains a set of available components that you can put in the registration form. The main menu of the form allows Inspector and Designed Form. The description of the components' properties can be seen below. To place a component in the form, click the necessary component on the Component Palette and then click on the Registration Form to put it in the necessary position.



There are the following controls available:

- | [TForm](#) - the registration form itself;
- |  [TEdit](#) control allows you to enter the registration name and key, shows the hardware ID or contains any user data;
- |  [TLabel](#) control allows you to show any text;
- |  [TGroupBox](#) control allows you to group several controls in the form and also may contain a header text;
- |  [TMemo](#) control is very similar to TEdit control, but may have multiple lines of the text;
- |  [TButton](#) is a push button control that allows managing user actions such as registering, closing the application, etc.
- |  [TSpeedButton](#) control is very similar to TButton, but may also contain an image;
- |  [TLink](#) control allows you to open a browser window, a default email client, some text file in the form, or to open/execute any other file;
- |  [TImage](#) control allows you to draw a simple geometric shape in the form;
- |  [TShape](#) control allows you to place an image in the form;
- |  [TBevel](#) control allows you to create beveled boxes, frames, or lines;
- |  [TPanel](#) control allows you to put an empty panel in the form. Panels have features for providing a beveled border, as well as methods to help manage the placement of child controls embedded in the panel;

File menu



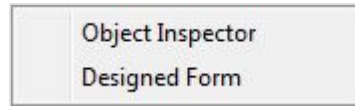
- | Load from File - allows you to load the registration form from an external file;
- | Save to File - allows you to save the current registration form to an external file;
- | Exit - exit the designer;

Edit menu



- | Cut - cut selected control(s) from clipboard;
- | Copy - copy selected control(s) to clipboard;
- | Paste - paste copied/cut control(s) from clipboard to the form;
- | Delete - delete selected control(s) from the form;
- | Select All - select all control(s) on the form;
- | Bring to Front - bring selected control(s) to the front;
- | Send to Back - send selected control(s) to the back;
- | Align to Grid - align selected control(s) to the grid;

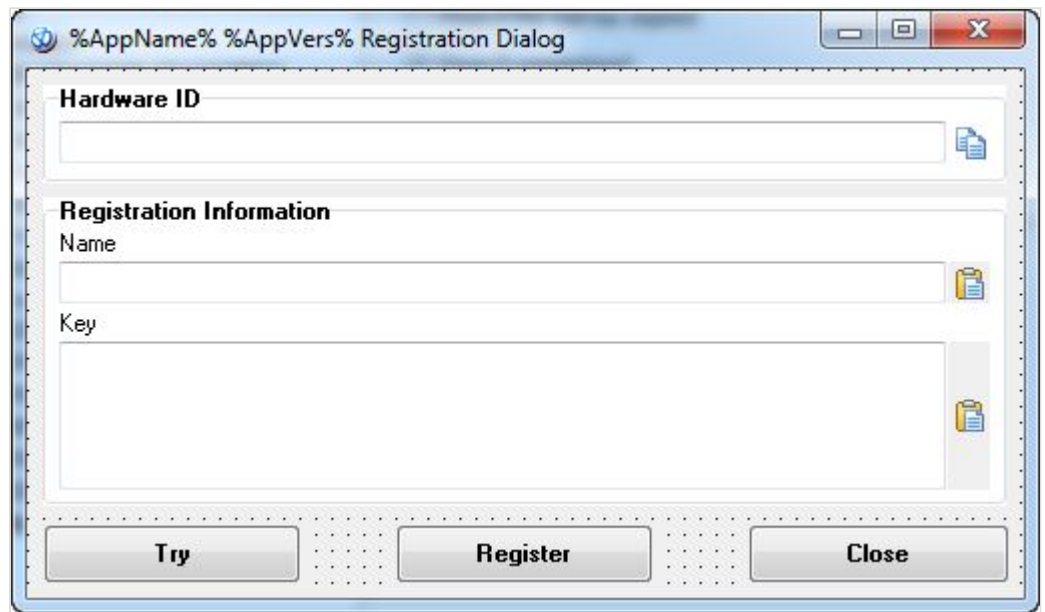
View menu



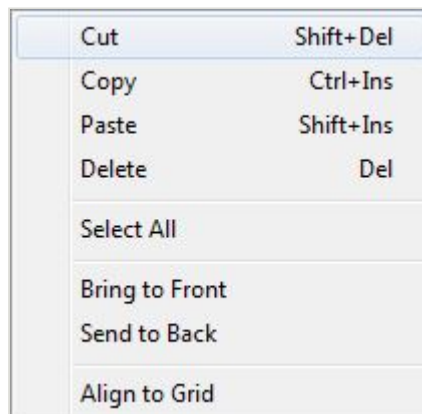
- | Object Inspector - shows Object Inspector;
- | Designed Form - shows the designed registration form;

Registration Form

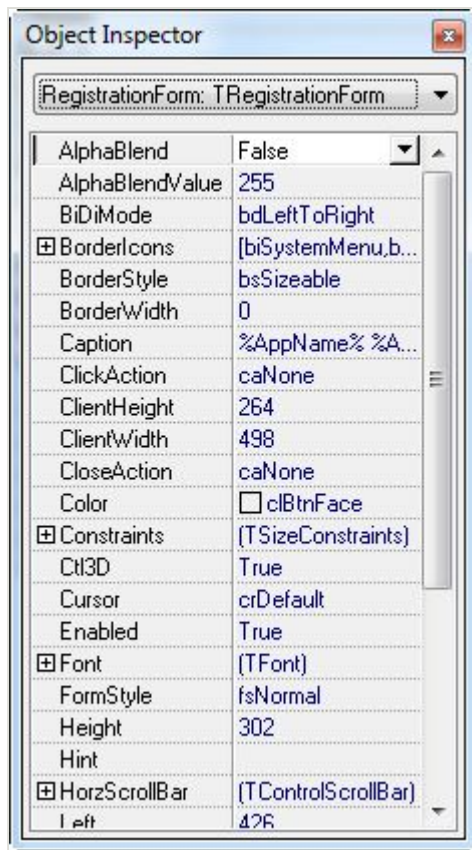
It is a form that will be shown to the users for them to enter registration information. Click on the necessary component form and its properties will be shown in the Object Inspector. To delete the component from the form, right-click the component and select the Delete item in the appeared pop-up menu, or press the Del key (Note: the registration form cannot be deleted).



Right-clicking shows the registration form pop-up menu.



- | Cut - cut selected control(s) from clipboard;
- | Copy - copy selected control(s) to clipboard;
- | Paste - paste copied/cut control(s) from clipboard to the form;



Click here to view TForm properties;



Click here to view TEdit properties;



Click here to view TLabel properties;



Click here to view TMemo properties;



Click here to view TGroupBox properties;



Click here to view TButton properties;



Click here to view TSpeedButton properties;



Click here to view TLink properties;



Click here to view TImage properties;



Click here to view TShape properties;

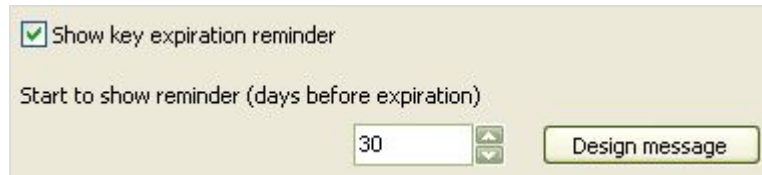


Click here to view TBevel properties;



Click here to view TPanel properties;

Key Expiration Reminder



The screenshot shows a configuration panel with a light beige background. At the top, there is a checked checkbox labeled "Show key expiration reminder". Below this, the text "Start to show reminder (days before expiration)" is followed by a numeric input field containing the value "30" and a small spinner control. To the right of the input field is a button labeled "Design message".

Allows you to notify the user that the registration will soon expire. The feature works only if the registration key has an expiration date.

Show Registration Key Reminder - enables the key expiration reminder message to be shown.

Start to show reminder (days before expiration) - specify the number of days before the key expiration when the user should get a reminder message. Click the "Design Message" button to design a reminder message, see [Message Designer](#) for more information.

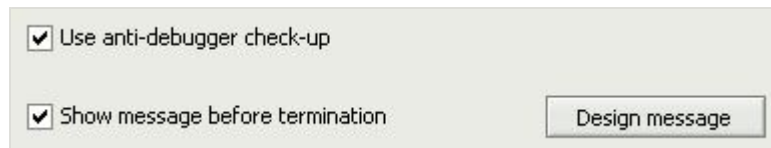
Check-Up

The functions of this topic allow adding a set of checkups like the checkup of file content changing, memory patching, etc.

Follow the links below to get more details.

- | [Anti Debugger](#)
- | [Control Sum](#)
- | [Startup Password](#)
- | [File Name](#)
- | [Disk Drive](#)
- | [Executed Copies](#)
- | [User Language](#)
- | [External Files](#)
- | [Executed Processes](#)
- | [Loaded Drivers](#)
- | [Installed Services](#)
- | [Windows Version](#)
- | [Virtualization Tools](#)
- | [Privileges](#)

Anti Debugger



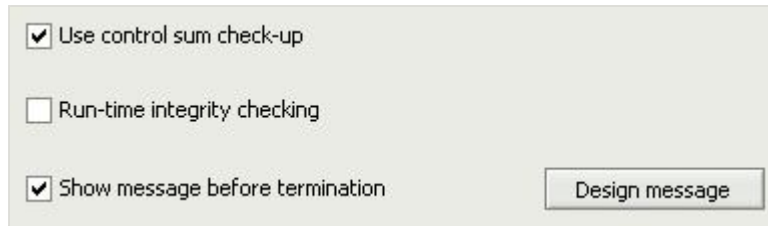
The image shows a settings dialog box with a light gray background. It contains two checked checkboxes: "Use anti-debugger check-up" and "Show message before termination". To the right of the second checkbox is a button labeled "Design message".

Use anti-debugger check-up - checks the debugger presence. A debugger is any tool which provides reversing of sources during the application work. The examples of debuggers are SoftIce, OllyDbg, TWD etc.

If a debugger is found while the module is running, the execution process will be terminated.

Show Message before termination - if this option is enabled a reminder message will be shown in case a debugger is found. To edit the content of the message, press "[Design Message](#)". In case the option is disabled, the execution of the module will be terminated without notification.

Control Sum



The image shows a dialog box with three checked checkboxes and one unchecked checkbox. The checked options are 'Use control sum check-up', 'Show message before termination', and 'Design message' (which is a button). The unchecked option is 'Run-time integrity checking'.

<input checked="" type="checkbox"/> Use control sum check-up	
<input type="checkbox"/> Run-time integrity checking	
<input checked="" type="checkbox"/> Show message before termination	<input type="button" value="Design message"/>

Use control sum check-up - serves for tracking module source codes changes. Module source codes changes refer to the changes of the file content, memory changes during the module start and work. The reasons of source codes changes can be cracks or viruses.

In case any modifications of the module file or the module memory are detected during the module start, the execution process will be terminated.

Run-time integrity checking - enables the technique of monitoring the memory source code modifications during the module work. Most executables have parts of the codes located in the memory with read only access characteristics, the current technique continuously watches the read-only memory parts and checks these control sums. In case any modification is detected, the process will be terminated immediately. The modifications of the memory parts with read-only characteristics can be caused only by the influence of crackers or viruses.

Show Message before termination - enable it if you need to notify the user about the module change and module termination. To edit the message, press "[Design Message](#)" button. In case the option is disabled, the execution of the module will be terminated without notification.

Startup Password

The screenshot shows a configuration window for startup passwords. It has a light beige background and is divided into several sections:

- Use password check-up:** A checked checkbox.
- Action:** Three radio buttons: "Always ask for password" (unselected), "Ask for password only first time" (unselected), and "Ask in days" (selected). To the right of "Ask in days" is a numeric input field containing "30" and up/down arrow buttons.
- Additional options:** Two checked checkboxes: "Hide password symbols" and "Allow to change password".
- Passwords:** A table with two columns: "Password" and "Commentary". It contains three rows of data:

Password	Commentary
pass	Password for Mr.1
pass1	Password for Mr.2
pass2	Password for Mr.3
- Buttons:** "Add", "Edit", and "Delete" buttons are located below the table. A "Design message" button is located at the bottom left of the dialog.

Use password checkup - enables startup password checkup. If this feature is enabled, the user will get a dialog on the file start that asks to enter a password. If the user enters the valid password, the execution will continue, otherwise a message will be shown and termination will take place. To design the message, click [Design Message](#) button. You can set any number of passwords and commentaries to each pass. A commentary is any string, it is used only as a reminder and does not affect module workability.

Always ask for password - always ask for a password when the module is executed;

Ask for password only for the first time - ask for a password every execution until a valid password is typed. After you have typed a valid password for the first time, it won't be asked anymore.

Ask in days - once a valid password is typed, the dialog will be shown again after a defined number of days.

Hide password symbols - if option is enabled, the password symbols will be hidden/replaced with "*" symbol. Otherwise, the password will not be hidden;

Allow to change password - allows the user to change the startup password. Note that this feature works for one password per PC only. For example, if the user uses 2 passwords on one PC and decides to change the first one, it will be possible; if the user decides to change the second one, they will succeed as well, but the first password will return to the original one.

To add, edit, delete the password, press the following buttons.

Password designer

Password:

Commentary:

Ok Cancel

File Name

Use file name check-up

Original file name:
notepad_protected.exe

Show message before termination

Use file name check-up - checks changes of the module file name. Use it to prevent changes of the module file name.

During the module start, the loader checks the module file name (which is placed in the "Original file name" field), in case it differs from the original one the module execution will be terminated.

"*Get Actual*" button - read the file name from "Output-File name of resulting program module" and place in the "Original file name" field.

Show Message before termination - enable it, if you need to notify the user that there is an error in the module file name. To edit the message, press "[Design Message](#)" button. If this option is disabled, the execution will be terminated without notification.

Disk Drive

Drive Type	Execution Permissions
Floppy disk (removable)	Deny
Hard Disk Drive	Allow
Network drive	Deny
CD-ROM drive	Allow
RAM disk	Deny

Use drive disk type check-up - serves for checking the drive type on which the module is executed. The function can be used in cases when you want to limit module execution depending on the drive type (hard disk, CD-Rom, etc).

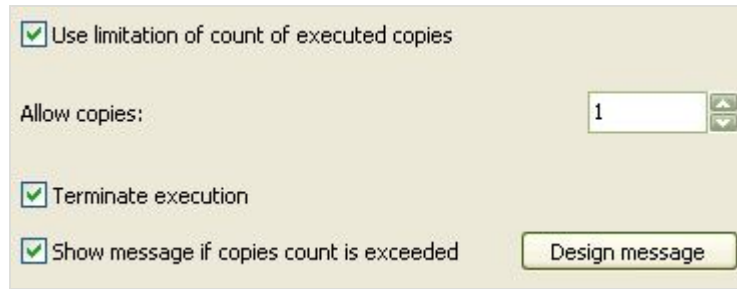
While the module is starting, the loader is checking the drive type on which the module is placed. If the current module type is denied by the loader, the execution will be terminated.

Selection:

- | Floppy disk (removable).
- | Hard Disk drive.
- | Network drive.
- | CD-ROM drive.
- | RAM disk.

Show Message before termination - if it is enabled, the user will be notified in case the module isn't executed due to the drive deny. To edit the message, press "[Design Message](#)" button. If this option is disabled, the execution will be terminated without notification.

Executed Copies



Use limitation of count of executed copies

Allow copies:

Terminate execution

Show message if copies count is exceeded

Design message

Use limitation of count of executed copies - serves for limiting the number of simultaneously executed module copies.

Allow copies - choose the possible count of simultaneously executed copies of the module.

Terminate Execution - with this option enabled, the execution will be terminated if the number of simultaneously executed copies of the protected application exceeds the defined value. This option may be very helpful for checking the copies count manually through the [EP_CheckupCopies](#) Enigma API.

Show message if copies count is exceeded - if the count of simultaneously executed copies of the module exceeds the value defined in the "Allow copies" field, a warning message will be shown and execution of the module will be stopped. To edit the warning message, press "[Design Message](#)" button.

While the module is starting, the Enigma loader will be checking how many copies of the current module have already been executed, if the module count exceeds the defined value, then execution will be stopped.

User Language



Lock execution to user language - allows you to lock execution of the protected module to a user country. Select possible countries to execute the module. If the module is executed in a non-selected country, an alert message will be shown (if the "Show message before termination" option is enabled) or the module will be terminated silently;

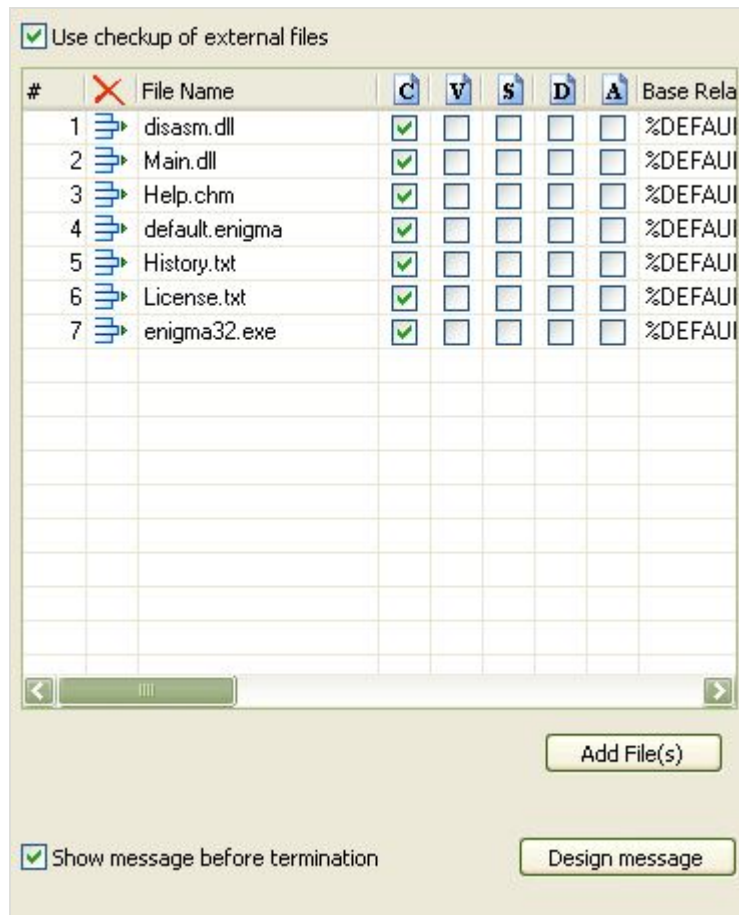
Select all - select all items in the countries list;

Unselect all - deselect all items in the countries list;

Invert selection - invert selection of all items, i.e. deselect all selected items and vice versa.







Show Message before termination - with this option enabled, the user will be notified in case the module isn't executed due to the user language lock. To edit the message, press "[Design Message](#)" button. If this option is disabled, the execution will be terminated without notification.

External Files



Use checkup of external files - checking external files. During the software distribution, it is a good idea to check the content of distributive files against changing. After adding the file into the list, define the necessary parameters to checkup. If at least one checkup parameter differs from the defined one while the protected file is starting, it will be terminated. Click on the files list,

column "Action" - you will see the list of available checkups for files, the common ones are:

- |  Delete file from the list
- |  File checksum - checks the full content of the file
- |  File Version - checks the version of the file (this option is for Win32 executable files only)
- |  File Size - checks the size of the file
- |  File Date - checks the date of the file
- |  File Attributes - checks attributes of the file.

column "Base Relative Path" - define a base relative path of the checkup file relatively to the main protected module.

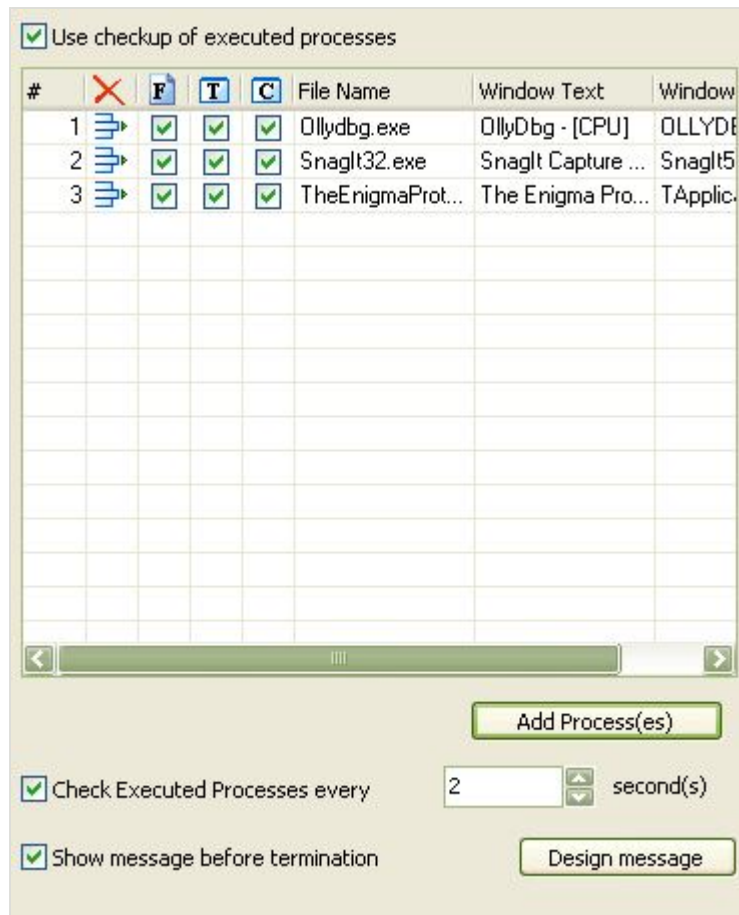
column "Relative Path" - define a relative path of the checkup file relatively to "Base Relative Path".

Example: you have the main protected module and you need a checkup of the file that is placed in the "Example" subfolder relatively to the main protected module, you need to add the file to the list, choose "Base Relative Path" to "%DEFAULT FOLDER%" then add the name of the subfolder "Example" in the "Relative Path".

Add File - press this button to add a file(s) into the list.

Show message before termination - if Enigma checks that the external file has been modified, it will show a message and will be terminated. To edit the warning message, press "[Design Message](#)" button.

Executed Processes



Use checkup of executed processes - checking executed processes. This feature allows you to make your own list of the "black" programs that can not be executed together with the protected module. There you may define a list of the software that should not be executed while the protected module is working. How does it work? While the protected file is working and starting, the Enigma loader is enumerating all the executed processes and checking if there is any software blocked by the function. If blocked software is found, the protected module is immediately terminated. For example, this feature allows you to get protection against OillyDbg (software debugger), SnagIt (screen capture tool), file and registry monitors. You can define checkup parameters (click on the columns to edit):

column "Action" - you will see the list of available checkups for the processes, the common ones are:

- 1. [X] Delete process from the list
- 2. [F] File Name - checkup of the file name of the executed process
- 3. [T] Window Text - checkup of the name of the process window (Warning: the window text may be changed while the application is working)
- 4. [C] Window Class - checkup of a window class.

column "File Name" - define a file name of the process to search.

column "Window Text" - the text of the window to search. If you want to search the window text by mask, you can enter the text quoted with an asterisk (*) symbol. For example, entered text *OillyDbg* will search any window which text (any part of text) contains OillyDbg word.

column "Window Class" - the class of the window to search.

Add Process - press this button to add a process(es) into the list. Note: you may add any process to the list and then edit it as you want. The appeared window shows you all the processes with their windows that are currently executed in your system.

Process	Window	File Name	Window Text	Window Class	Current F
00000400	00010076	Explorer.EXE		tooltips_class32	C:\WIND
00000400	00010066	Explorer.EXE		tooltips_class32	C:\WIND
00000400	00070508	Explorer.EXE		WorkerW	C:\WIND
00000400	0002004A	Explorer.EXE		tooltips_class32	C:\WIND
00000400	00020080	Explorer.EXE	Start Menu	DV2ControlHost	C:\WIND
00000400	0001007C	Explorer.EXE		tooltips_class32	C:\WIND
00000400	0001007A	Explorer.EXE	CiceroUIWndFra...	CiceroUIWndFra...	C:\WIND
00000400	00010068	Explorer.EXE		tooltips_class32	C:\WIND
00000400	0003004C	Explorer.EXE		tooltips_class32	C:\WIND
00000400	00020064	Explorer.EXE		tooltips_class32	C:\WIND
00000400	0003003A	Explorer.EXE		Shell_TrayWnd	C:\WIND
00000400	00020044	Explorer.EXE		tooltips_class32	C:\WIND
00000400	00030502	Explorer.EXE		BaseBar	C:\WIND
00000400	0005050A	Explorer.EXE		BaseBar	C:\WIND
00000400	00020082	Explorer.EXE		tooltips_class32	C:\WIND
00000400	0001008C	Explorer.EXE		tooltips_class32	C:\WIND
00000400	0002052C	Explorer.EXE		tooltips_class32	C:\WIND
00000400	000400FC	Explorer.EXE	MS_Webcheck...	MS_Webcheck...	C:\WIND
00000400	00010184	Explorer.EXE	Connections Tray	Connections Tray	C:\WIND
00000400	000B0136	Explorer.EXE	Power Meter	SystemTray_Main	C:\WIND

Check Executed Processes every X second(s) - it enables runtime checkup of executed processes while the protected application is working. Set up the optimal number of seconds delay between the checkups (if you have many items, for example, greater than 50, to checkup in the list, we recommend setting number of seconds greater than 10 to avoid high CPU loading on slow PCs). If this feature is disabled, the Enigma will check executed processes only once upon the application startup.

Show message before termination - if the Enigma checks that there is an executed process that matches the criteria, it will show a message and will be terminated. To edit the warning message, press "[Design Message](#)" button.

Loaded Drivers

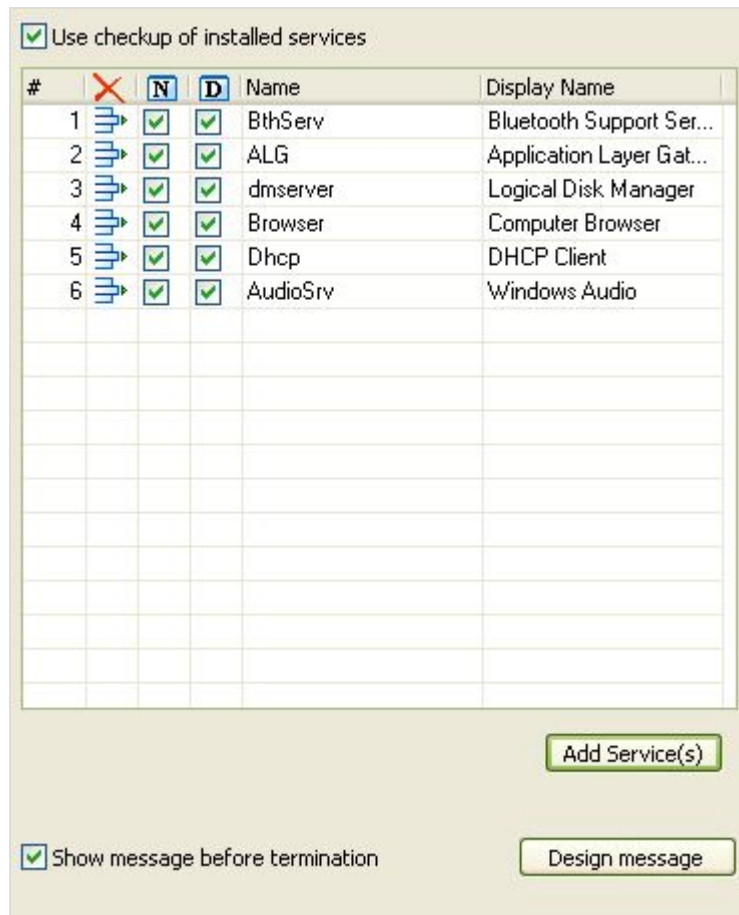


Use checkup of loaded drivers - it checks if the drivers with the defined names are installed in the system. A lot of reversing tools use drivers to perform some work, use this checkup to detect hack tools and avoid execution of the protected file. This checkup works not only at the file start, but also all the time during the work of the protected file.

To add a driver to the list, click the Add button and edit the Driver Name column. To edit the driver name, click on the necessary row at the Driver Name column and edit the text. To delete a driver from the list, click image at the row to delete.

Show message before termination - if Enigma detects that the external file has been modified, Enigma will show a message and will be terminated. To edit the warning message, press "[Design Message](#)" button.

Installed Services



Use checkup of installed services - checks the installed services. Warning: the feature does NOT support Windows 95, 98 and ME and requires Administrator privileges. It allows you to make your own list of the "black" services that could not be installed together with the protected module. You can define checkup parameters (click on the columns to edit):

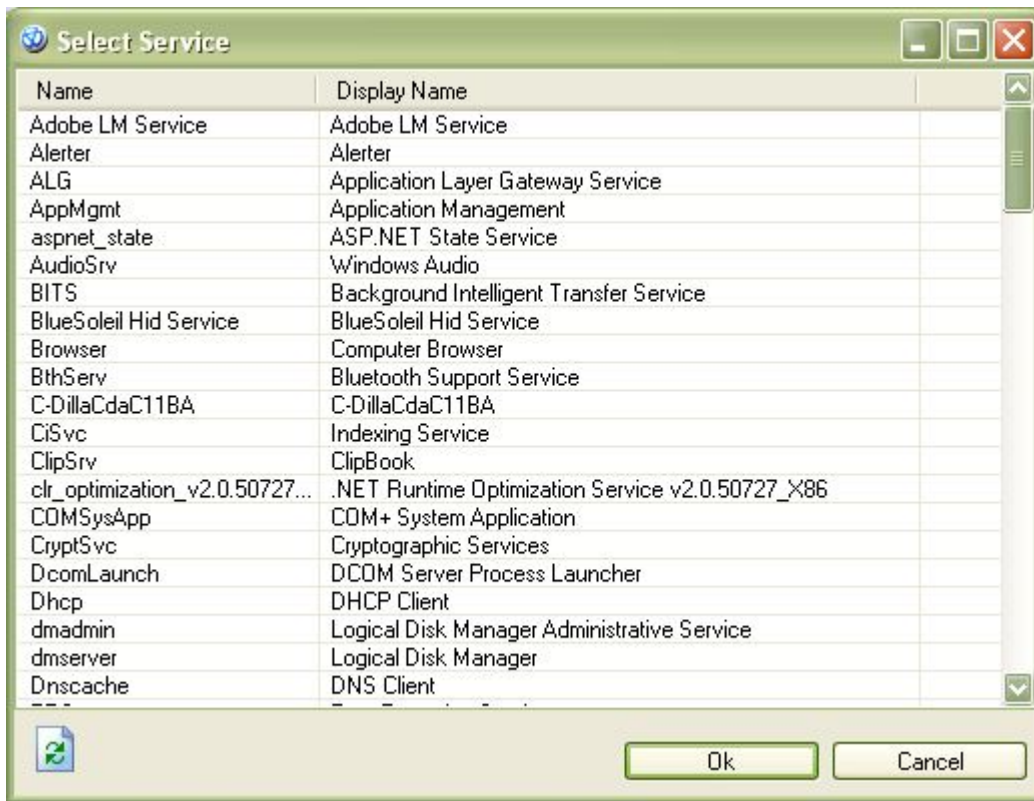
column "Action" - you will see the list of available checkups for the services, the common ones are:

- Delete service from the list
- Name - the service name
- Display Name - an expanded service name.

column "Name" - define the name of the service to search.

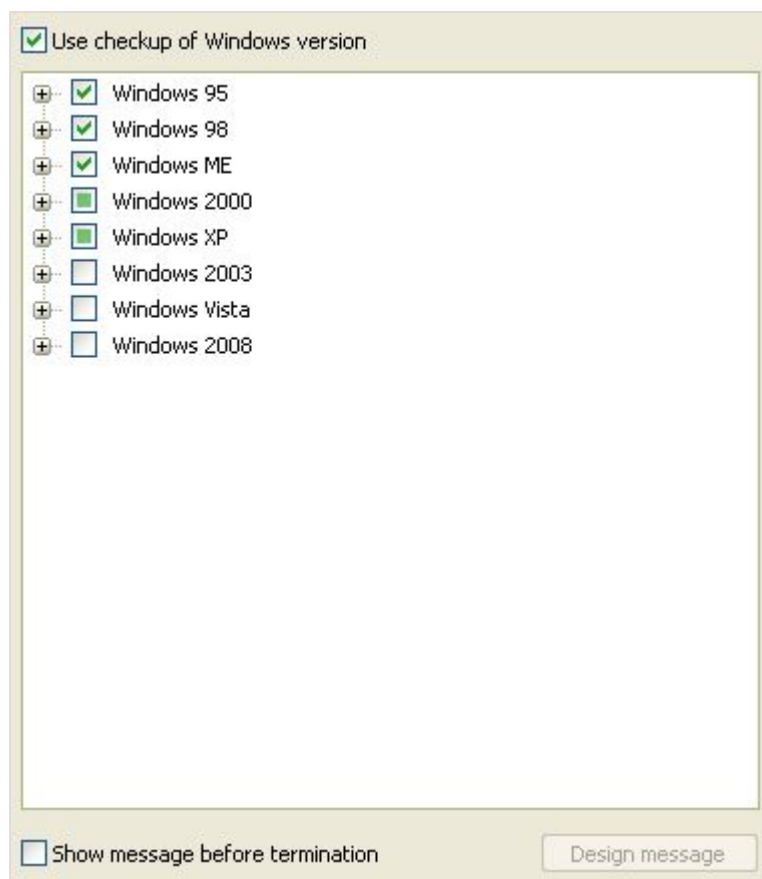
column "Display Name" - an expanded service name.

Add Service - press this button to add a service(s) into the list. Note: you may add any service to the list and then edit it as you want. The appeared window shows you all the services that are currently installed in your system.



Show message before termination - if the Enigma detects that there is a service that matches the criteria installed, it will show a message and will be terminated. To edit the warning message, press "[Design Message](#)" button.

Windows Version

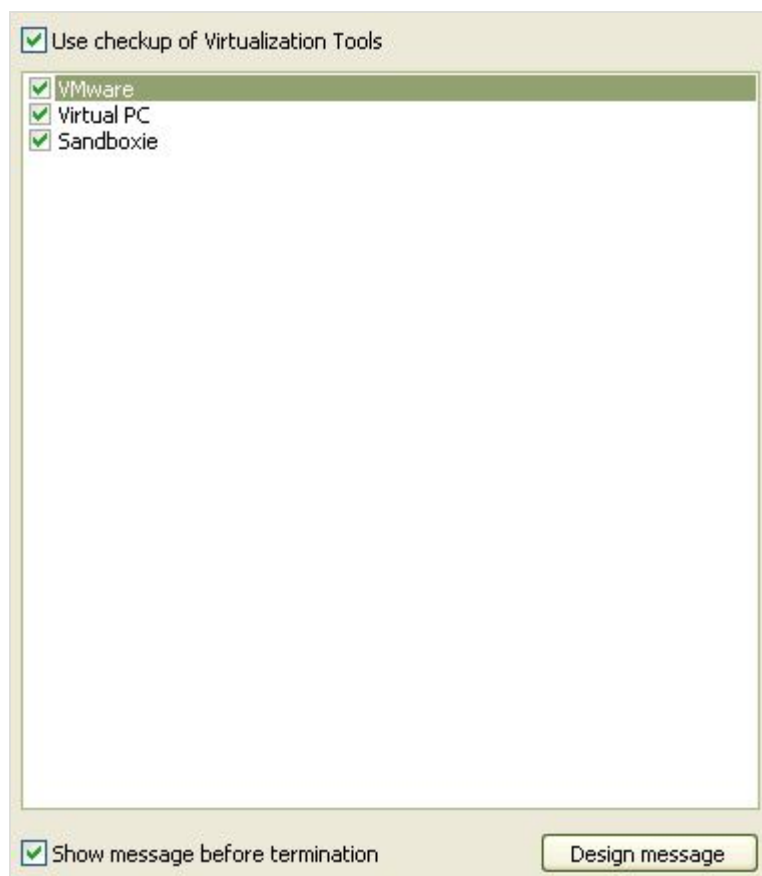


Use checkup of windows version - checks the Windows version. It allows you to DENY execution of the protected module for a particular Windows version. If the Enigma loader detects during the execution that the current Windows version is denied, then the protected module will be terminated.

If you need to add more Windows versions, please, contact our support team!

Show message before termination - if the Enigma detects that the current Windows version is denied, then a message will be shown and termination will take place. To edit the warning message, press "[Design Message](#)" button.

Virtualization Tools



Use checkup of Virtualization Tools - checks if the file is executed under Virtual Machines. If the file runs under the Virtual Machine, the execution is terminated.

If you need to add some more Virtual Machines to the list, please, contact our support team!

Terminate Execution - if this option is enabled, the Enigma Protector will terminate execution of the protected file in case virtualization tools have been found. It is recommended to uncheck this option if you plan to use Enigma API function [EP_CheckupVirtualizationTools](#).

Show message before termination - if the Enigma detects that the file runs under a Virtual Machine, then a message will be shown and termination will take place. To edit the warning message, press "[Design Message](#)" button.

Privileges



Some applications (or some features in the application) require high user's privileges. Windows NT systems support (basically) two types of user accounts: administrator and limited.

Use checkup of administrator privileges - serves to check if the user's account under which the protected module is executed has administrator privileges or not. If the account is limited, then the protected application will not be started. Note: for Windows ME, Windows 98 and Windows 95 it always returns true (meaning that the user has already got high privileges).

Show message before termination - if you want to inform the user that they should increase their user privileges, enable this option. To edit the message, press "[Design Message](#)" button. If this option is disabled, the execution of the module will be terminated without notification.

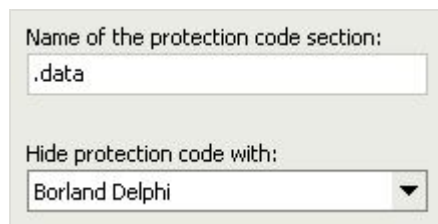
Protection Features

The topic describes the possibilities for adding some functions to make the protected module more resistant to reversing of source codes.

Follow the links below to get more details.

- | [File Analyzer Deception](#)
- | [Original File Size Preservation](#)
- | [Extra Resource Protection](#)
- | [Advance Force Import Protection](#)
- | [WinAPI Redirection](#)
- | [WinAPI Emulation](#)
- | [Inline Patching](#)
- | [Protected Strings](#)
- | [Resources Protection](#)

File Analyzer Deception



The image shows a configuration dialog box with two fields. The first field is labeled "Name of the protection code section:" and contains the text ".data". The second field is labeled "Hide protection code with:" and is a dropdown menu currently showing "Borland Delphi".

It provides hiding the protector signature from file analyzers, such as PEiD.

The Enigma Protector inserts some simple part of the code in the module entry point that allows you to deceive file analyzers.

Name of the protection code section - the name of the section where the loader code will be placed. The section name is a string of 8 characters length.

Hide protection code with - the type of deception. It defines the code that will be used for analyzer deception. Select the compiler type that will be determined in analyzers. If there is no need to use deception, set this option to "None".

Warning: the use of the "Microsoft Visual C++" deception type can cause unexpected errors for modules written in Visual Basic.

Original File Size Preservation

Use original file size preservation

Use original file size preservation - allows you to save the size of the protected module the same as the original one. This helps to hide the protector presence from new-come crackers.

Remark: if the protected module has a larger size than the original one, the options will be ignored.

The Enigma Protector just rewrites the file protected with a set of bytes to make its size equal to the size of the original file.

Advanced Force Import Protection

Use advance force import protection

Use advanced force import protection - advanced protection of the import table. It allows you to delete the import table of the protected module in the memory. The Enigma Protector searches all entries in the import table in the source code, and changes the direct link to the imported functions.

WinAPI Redirection

Use WinAPI functions redirection

It provides redirection of calls to system libraries.

After the module start, the loader checks the content of system libraries and redirects calls from the import table of the module protected to WinAPI. Also, part of system WinAPI functions are emulated, i.e. a set of instructions from WinAPI functions is obfuscated without loss of workability.

WinAPI Emulation

Use WinAPI functions emulation

Redirects some kinds of system WinAPI functions to emulated internal Enigma functions. Internal Enigma functions fully emulate the work of system WinAPI functions, so it should not affect the workability of the module protected.

Please note that if you use Enigma API inside your application, this function should be enabled.

Inline Patching



Use Inline Patching Protection

Options

Number of protection threads: 3

Inline checkup delay: 1000

Allows you to apply an Anti-Inline Patching technology. The Anti-Inline Patching technology uses an advanced method that checks the integrity of the protection code. If any changes of the protection code are detected, the file execution will be immediately terminated without any prompt.

How it works: when the protected file starts, the loader executes some threads (the number of threads may be defined manually) and each thread periodically checks the integrity of the protected code (the delay between the checkups of each thread is defined in "Inline checkup delay" field).

Number of protection threads - define the number of protection threads. The optimal value is 3-5 threads. We do not recommend using just one, but the maximum number is 15. Please note that the high number of threads may cause high CPU loading. You may select the necessary number of threads after the complete testing of the file protected.

Inline checkup delay - define the number of milliseconds between every checkup of each protected thread. We do not recommend setting this number to a high value, because it may not be useful and can be bypassed, but a too small value may cause high CPU loading, too. Test the protected file and select the optimal value.

The Protected String might be defined by two types of values:

- ID - a unique integer value;
- Key - any unique string.

You may change the ID and Key, but remember that the ID and Key should be unique for each protected string in the list. The Enigma Protector generates its own unique ID and Key for each string that you add, you may keep self-generating values.

To get a protected string by its ID, use the [EP_ProtectedStringByID](#) Enigma API function or [EP_ProtectedStringByKey](#) to get it by its Key value.

Resources Protection


Select resources protection type

Protect all resources

Protect all resources, except icon

Protect all resources, except icon and version

Resources Protection allows you to select the types of resources in the executable file that will be protected. All resources in the file are protected by default, but there you can enable/disable additional protection of ICON and VERSION resources.

- Protect all resources - all resources in the file will be protected;
- Protect all resources, except icon - do not protect ICON resources. Almost all executable files contain ICON resources, a default icon is displayed with the file shown in the file Explorer. Note that if the "Protect all resources" option is enabled, the default Windows icon  will be shown for this file in Explorer.
- Protect all resources, except icon and version - disables protection of ICON and VERSION resources. Note that if any of the two above options are enabled, meaning that VERSION resources are protected, then the file version/information will not be shown in the properties dialog. The current option is recommended for most files.

Use the first two options only if you definitely need to protect ICON and VERSION resources.

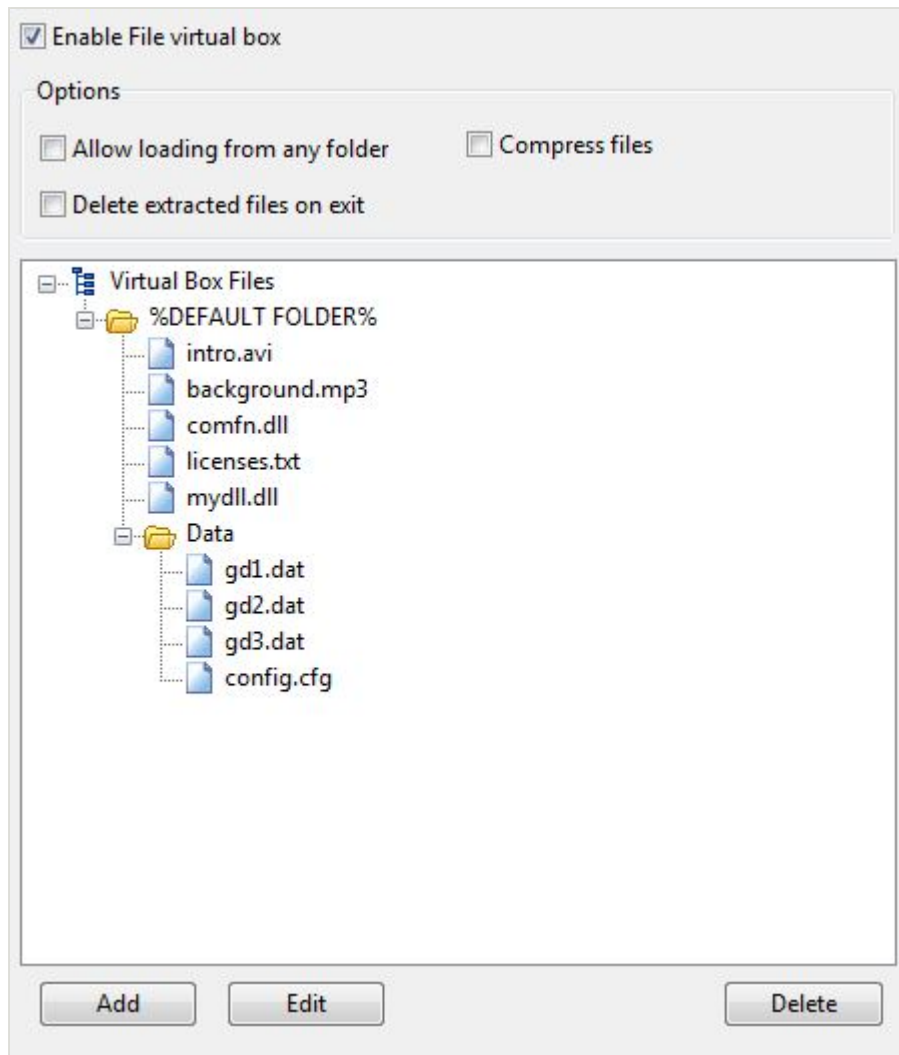
Note that the following resources are not protected either: STRINGS, MANIFEST, 'TYPELIB', 'REGISTRY'.

Virtual Box

Virtual Box contains additional features that allow you to emulate the file system of the protected application. You can select the necessary files that will be embedded into the protected application and use it without having these files on the disk. Follow the link below to read more:

- | [Files Virtual Box](#)

Files



Enable Files virtual box - an enabled Virtual Box feature for files. The Files Virtual Box allow you to embed any kind of external files that your application uses for a single executable file. After the protection, you may delete all the files listed in Virtual Box, the application will work without them. Virtual Box supports any kind of embedded files, txt, dll (dll files will be correctly processed and initialized when the application requires them, ActiveX files will be registered at the file start), avi, mp3, bpl, ocx and many others.

Warning: it is not recommended to use the Virtual Box feature for the protection of the Dynamic Link Libraries (DLL) files. Only the DLLs called once at the start of the main executable file and never freed while the application is working can be protected with Virtual Box. The DLL files called/freed multiple times per one session should not be used in Virtual Box, because it may cause application crash.

Allow loading from any folder - embedded files can be sometimes called at an additional path that might not be specified upon the protection. If you enable this option, the Enigma loader will ignore the file's path.

Delete extracted files on exit - this works only with the files that have "Always Write to Disk" or "Write if not Present" actions. When the protected module is terminated, the Enigma loader will attempt to delete the extracted files from the disk.

Compress Files - enables compression of Virtual Box files. It is not recommended to use this option because it requires more memory to execute, increases execution time and decreases the protection of the Virtual Box files. The advantage is that it allows you to decrease the size of the file protected.

Click the Add button to add the files or folders into the list. There are the following adding functions available (all functions are also available through the right-click pop-up menu):

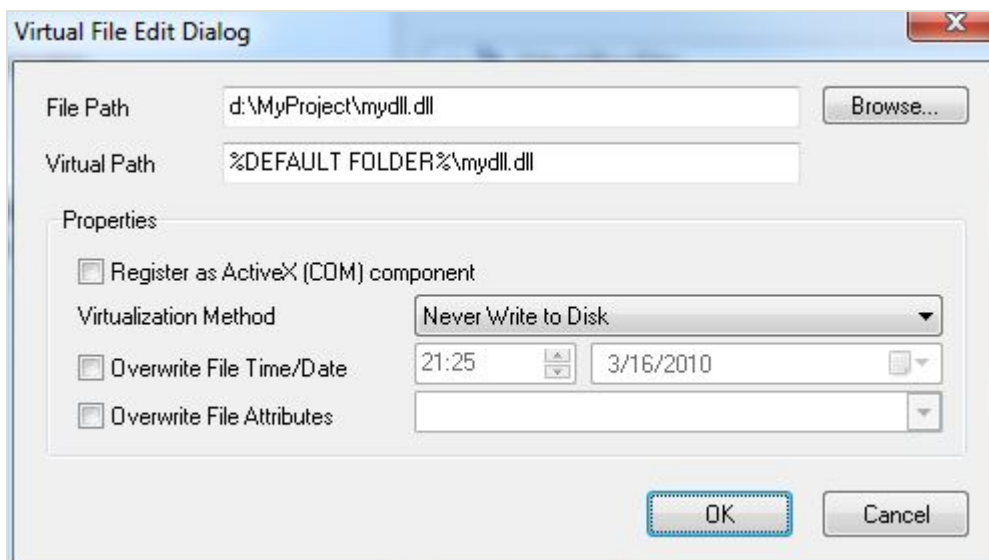
- 1 Add File (s) - allows you to add a file or several files to the list. The files will be added to the currently selected folder in the list;
- 1 Add Files Recursive - allows you to add a folder with the files and sub folders. Each subfolder will be

processed recursively;

- 1 New Folder - allows you to create a new folder in the list. Note that folders of the first root node can only be from a pre-defined list of folders or a root folder of the drive. It is not recommended to use absolute paths (the absolute path is defined through the drive letter in the root of the list), use them only in cases when the relative path could not be used. Read more about pre-defined folders below.

To delete a file or a folder, select it and click the Delete button or press the Del key.

To Edit the file properties or folder name, select it and click the Edit button, or double-click the necessary item. Files Edit dialog looks like the one at the screen below and has the following options.



File Path - points to the real file that will be embedded. This file must exist. Click the Browse button to change the file.

Virtual Path - specifies the position of the file in the virtual files list. Note that a virtual path can be started from a pre-defined folder (relative path) or from a drive letter (absolute path). There are the following pre-defined folder names that can be used for a start of the Virtual Path:

- 1 %DEFAULT FOLDER% - the folder where the protected module is placed;
- 1 %SYSTEM FOLDER% - System32 (WinNt) or System (Win9X) subfolder of the Windows installation folder;
- 1 %WINDOWS FOLDER% - the Windows installation folder;
- 1 %My Documents FOLDER% - My Documents folder. Attention: For operating systems of Windows NT family there is an individual "My Documents" folder for each user;
- 1 %My Pictures FOLDER% - My Pictures folder. It has the same warning as for My Documents folder;
- 1 %Program Files FOLDER% - Program Files folder;
- 1 %Program Files\Common FOLDER% - Program Files\Common folder;
- 1 %AllUsers\Documents FOLDER% - All Users\Documents folder;
- 1 %History FOLDER% - History folder. It has the same warning as for My Documents folder;
- 1 %Cookies FOLDER% - Cookies folder. It has the same warning as for My Documents folder;
- 1 %InternetCache FOLDER% - InternetCache folder. It has the same warning as for My Documents folder;
- 1 %ApplicationData FOLDER% - ApplicationData folder. It has the same warning as for My Documents folder.

Examples:

- 1 %DEFAULT FOLDER%\mydll.dll, means mydll.dll will be placed in the same folder with the protected file;
- 1 %DEFAULT FOLDER%\data\intro.avi, the file will be placed in the data folder that is located in the same folder with the protected file;
- 1 %SYSTEM FOLDER%\key.dat - the file will be placed in the system folder with name key.dat (for example, into folder c:\Windows\System32\)
- 1 C:\myfile.key, it is an absolute path where the file will be located.

Register as ActiveX (COM) component - if the option is enabled, Enigma will try to register this file as ActiveX (COM) library at the file start.

Virtualization Method - specifies how the file will be processed: it will be either extracted to the disk when the

protected file starts, or fully emulated in the memory. There are the following options available :

- | Never Write to Disk- the embedded file will never be written to the disk. How it works: the Enigma loader hooks file access Windows API and checks if the API call points to the embedded file. You do not need to do any programming in order to use this function. If you have already got an application, just embed the necessary file and protect it. Moreover, if the embedded file is a dll, you may call any functions from this library without having it on the disk. The Enigma loader will initialize dll in the memory and successfully call the function required. If you use this feature, you do not need to place the embedded files with the protected module. Using this feature brings you a lot of benefits: your external library will not be stolen and used as a third party software, you protect embedded libraries from cracking and modifying, you may hide any types of files such as MP3, AVI, any image files, etc;
- | Always Write to Disk - the embedded file will always be overwritten to the disk when the protected file starts;
- | Write if not Present - the embedded file will be written to the disk if such a file does not exist.

Overwrite File Date/Time - enter the date and the time values that will be applied to the embedded file. If the option is not checked, the time settings of the real file will be used.

Overwrite File Attributes - specify the attributed file that will be applied to the protected file. If the option is not checked, attributes of the real file will be used.

Virtual Machine

Virtual Machine is a modern virtualization technology that allows you to protect parts of the assembler source code of your module. The main idea of virtualization is converting the original assembler code (which is well-known to reverse engineers) to the PCODE - a special programming language known only to the Enigma Protector. When a protected application requires to run a virtualized code, the PCODE will be run on the internal Virtual Processor. Virtual Machine is a very useful feature that allows you to make reverse engineering/ analysing of the protected module very complex. Our recommendation is to use the virtualization technology as often as possible. Also, note that not all code parts/functions should be virtualized. If the part of the code or function is executed too many times, it is not recommended to virtualize it, otherwise it will slow down the work of your application and increase CPU loading. The best places for virtualization are parts of verifying registration code, trial control routines, cryptographic algorithms, in short, all weak places that should not be reversed/cracked/analysed.

There are several ways the virtualization technology could be implemented:

- | by means of [VM Markers](#). If you are a software developer, have the sources code of your application and can compile it, the best and most recommended way to apply virtualization is the VM Markers;
- | by using a MAP file (the MAP file contains a list of functions with their addresses, it has the same name as an input file choosen on [Input Panel](#) but has a .map extension. A MAP file allows accurate analysing of the input file), selecting the necessary functions. This is also a solution for software developers who can compile modules and generate MAP files. To learn more about the creation of MAP files for different development IDEs, see [Making Of the Map File](#). Note that a MAP file should be placed in the same folder as an input file and it should have the same time span as the input file name. See [Functions Selecting](#);
- | the third way, if you are not a software developer and cannot use the above two methods, you may simply click the Select Functions button and The Enigma Protector will try to analyse the input file and find all functions that are possible to be virtualized. Remember, this option is not so stable as the previous ones, so you have to make a double check of workability of the protected modules. See [Functions Selecting](#).

Limitations:

- | Virtual Machine does not work with any kind of .NET files, either exe or dll files;
- | if the part of code/functions selected for virtualization contains any [Markers](#) inside, the workability of the marker will be lost, the code inside the marker will only be executed;
- | for Delphi developers, the MAP file compiled with Delphi may contain a set of functions with the same names but different addresses. The Enigma Protector stores the selected functions by the name in the project file. After loading the project file, only the function with necessary name that appeared first will be selected;

Follow the links below for more information.

- | [File Entry Point](#)
- | [Functions Selecting](#)

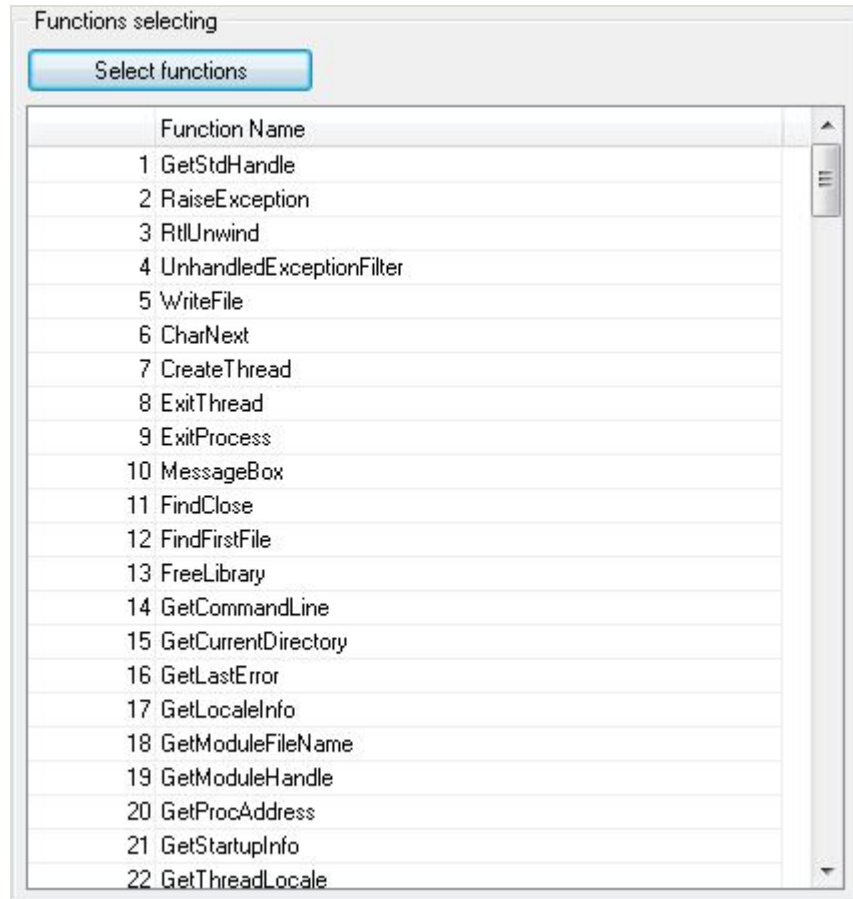
File Entry Point

Enable virtualization of file entry point

File Entry Point is a special function that the execution of any executable file starts with. The file entry point function runs only once per execution before any other code is executed.

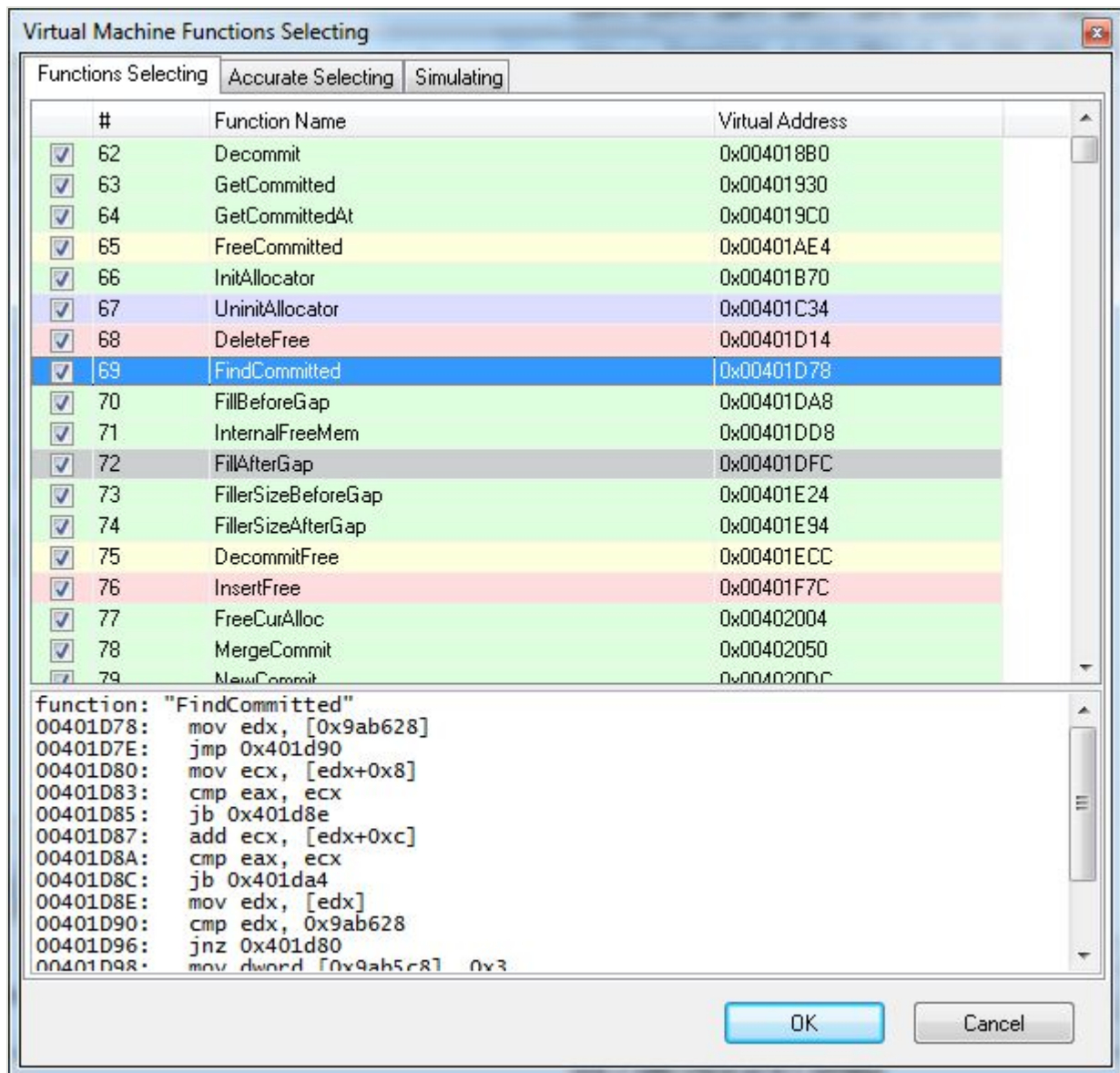
Enable virtualization of file entry point - enables virtualization of the file entry point. It is always recommended to use this feature.

Functions Selecting

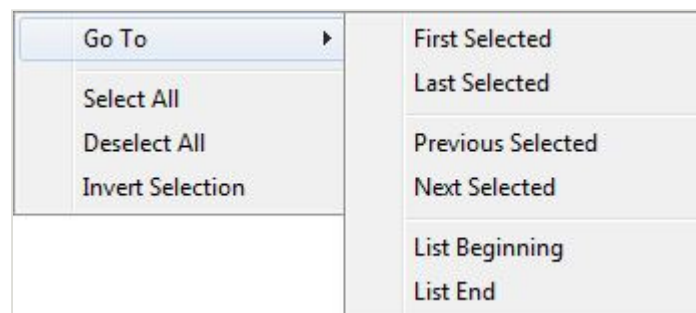


There is a list of functions names selected for virtualization. To change the list, click the Select Functions button. The below windows with the possibility to select functions should appear. Note that if you have a MAP file for the Input file, the list of functions will be shown, otherwise, Enigma will try to analyse the file itself and the functions names will be represented with their addresses.

Selection of Functions

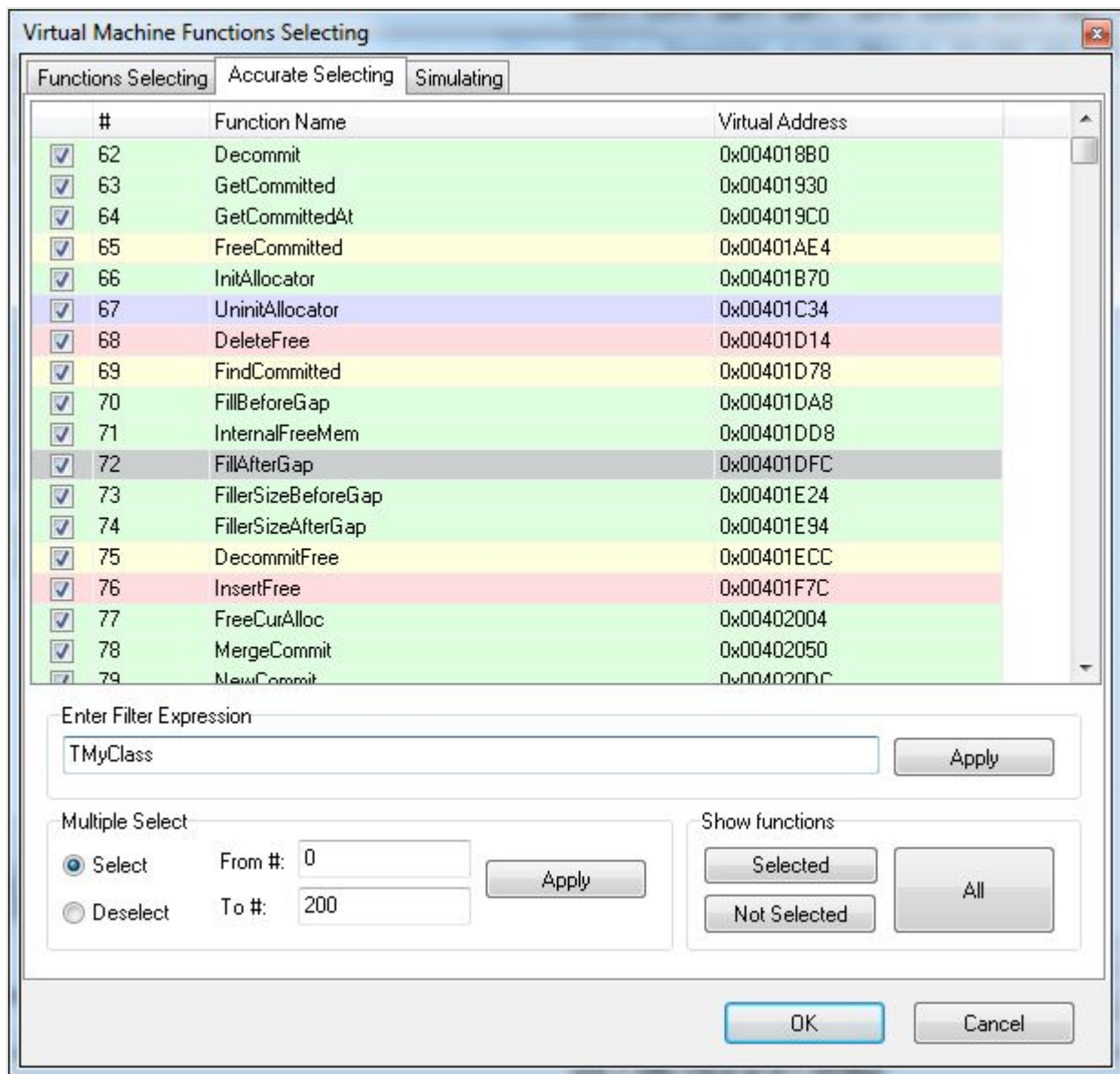


Select the functions necessary for virtualization by checking them out. The information field at the bottom shows a disassembled listing of the selected functions. As for the function in color, see the Simulating topic below. Right-click on the list to see the pop-up menu.



- | Go To - First Selected - allows you to scroll the list to the first checked function;
- | Go To - Last Selected - allows you to scroll the list to the last checked function;
- | Go To - Previous Selected - selects the previous checked function;
- | Go To - Next Selected - selects the next checked function;
- | Go To - Beginning List - scrolls the list to the beginning;
- | Go To - End List - scrolls the list to the end;
- | Select All - checks all functions in the list;
- | Deselect All - unchecks all functions in the list;
- | Invert Selection - checks the unchecked and unchecks the checked functions;

Accurate Selection



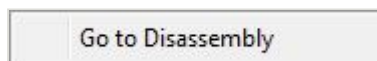
Accurate selection provides tools for easier navigation through the list of functions and their selection.

Enter Filter Expression - enter the text to sort the selected functions by. Only those functions that contain the text entered will be shown after you click the Apply button. This tool is very useful when the members of a particular class only should be selected. Just enter the class name, click the Apply button and only the members of this class will be shown.

Multiple Select - allows you to select and deselect a range of functions.

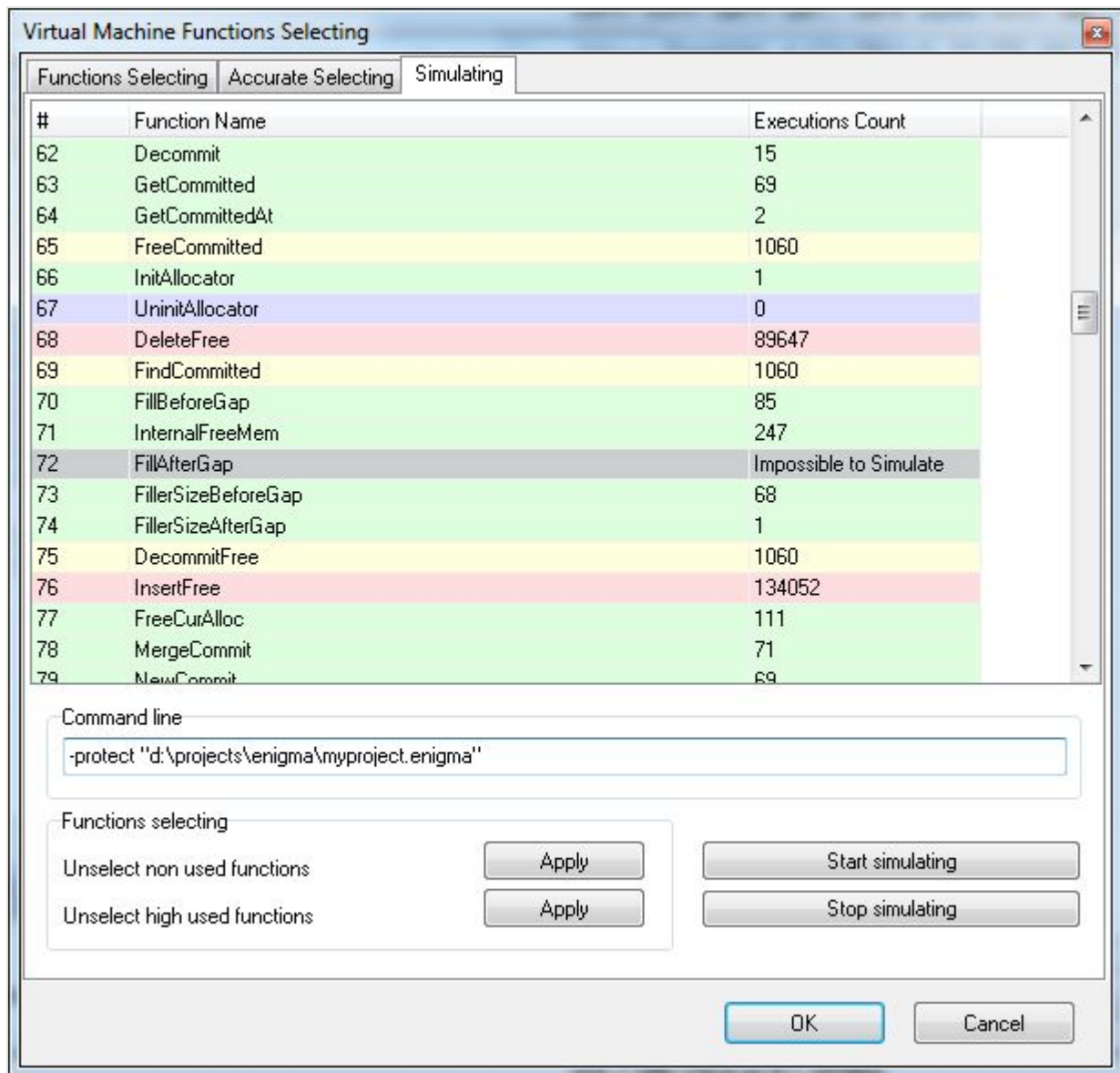
Show Functions - allows you to show only Selected or Not Selected functions, or both.

Right-click on the list to see the menu.



- Go to Disassembly - allows quick navigation to the Functions Selecting tab, for viewing a disassembled listing of the selected functions

Simulating



Simulating allows you to estimate how many times the selected functions have been executed in your application. While simulating, The Enigma Protector executes input file entering in [Input Panel](#), inserts some counters into the executed processes that allow you to count the number of times that a particular function has been executed. The Enigma Protector reads the counters each second while the application is working.

Click the Start Simulating button to start the simulating process. To stop simulating, close the executed application or click the Stop Simulating button. If your application requires the Command Line for some purposes, enter it.

Note that this feature does not work with dll (COM/ActiveX etc) files, only exe files are supported.

After simulating you may uncheck the highly used or unused functions. But unselecting these functions is not a requirement, but is just a recommendation. If you think the functions should be virtualized, there is no difference how many times they are executed while simulating. Also note that virtualization of highly used functions may slow down work of your application and increase CPU loading, and virtualization of non-used functions may also be dangerous because there is no way to check out the workability of the selected functions after protection.

Functions in color:

- | Grey color - The Enigma Protector is not able to simulate execution of this function and to count how many times it has been executed, or there were still no attempts of simulating;
- | Blue color - the function was not executed during the last simulation;
- | Green color - the number of times the functions were executed during the last simulation is above zero and below the highly used value. Such functions are ideal for virtualization;
- | Yellow color - the number of executions of the functions is near to the highly used one. You will have to check out the working speed of your application after protection;
- | Red color - the function is highly used, it is not recommended to select this function, because it will slow down the working speed of protected application.

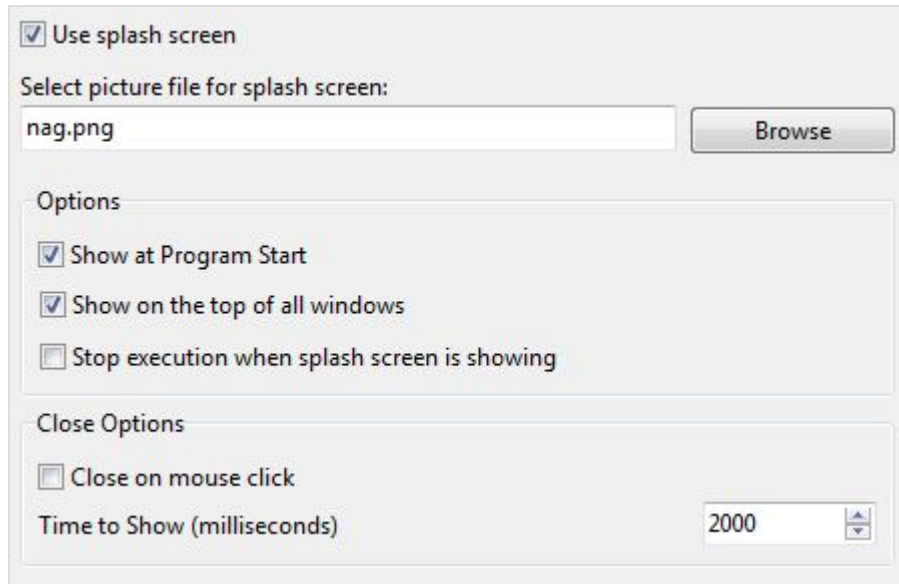
Miscellaneous

The topic contains a set of miscellaneous possibilities for module protection.

Follow the links below to get more details.

- | [Splash Screen](#)
- | [Watermark](#)
- | [Plugins](#)
- | [Custom VERSION Resource](#)
- | [Custom MANIFEST Resource](#)
- | [Markers Analyses](#)
- | [Command Line](#)
- | [Environment Variables](#)

Splash Screen



The screenshot shows a configuration dialog box for the splash screen. At the top, there is a checked checkbox labeled "Use splash screen". Below it, the text "Select picture file for splash screen:" is followed by a text input field containing "nag.png" and a "Browse" button. The dialog is divided into three sections: "Options", "Close Options", and "Time to Show (milliseconds)".

Options

- Show at Program Start
- Show on the top of all windows
- Stop execution when splash screen is showing

Close Options

- Close on mouse click

Time to Show (milliseconds) 2000

Allows adding a splash screen to the protected module. Splash screen may be just a picture that will be shown upon the startup of the protected file.

Select picture file for splash screen - the file name of the splash screen picture. The Enigma Protector supports 3 kinds of picture files: BMP, PNG, JPG (JPE, JPEG).

Show at Program Start - the splash screen will be shown at the program start, with this option checked. If you want to show this screen manually by means of Enigma API, then uncheck this option. See also Enigma API [EP_SplashScreenShow](#) and [EP_SplashScreenHide](#).

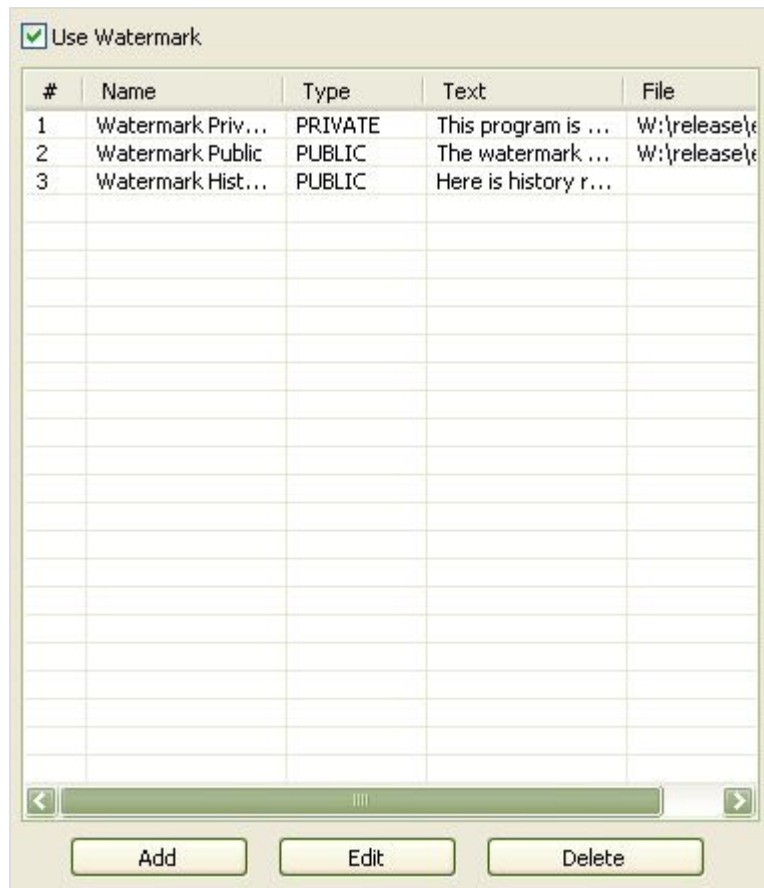
Show on the top of all windows - moves the splash screen onto the top of all windows.

Stop execution when splash screen is shown - execution of the protected file will be stopped when the splash screen is shown. If this option is disabled, then the protected file will continue to execute when the splash screen is shown.

Close on mouse click - the splash screen will be closed only upon being clicked.

Splash screen time to show (milliseconds) - the time during which the splash screen will be displayed. The time is in milliseconds.

Watermark



Use Watermark - allows you to add watermarks to the protected file. A watermark is just some information that is placed into the protected file. The watermark cannot be edited or modified in the file. If the watermark information stored in the file is corrupted, the file will not start. The Enigma Protector allows to place text/files watermarks. This feature might help developers to limit each version of the application to a particular user and then monitor if the version is shared and who shares it. The watermark cannot be deleted/modified in the protected file. See also, *Watermark Viewer*.

To Add/Edit/Delete watermarks, use the following buttons.



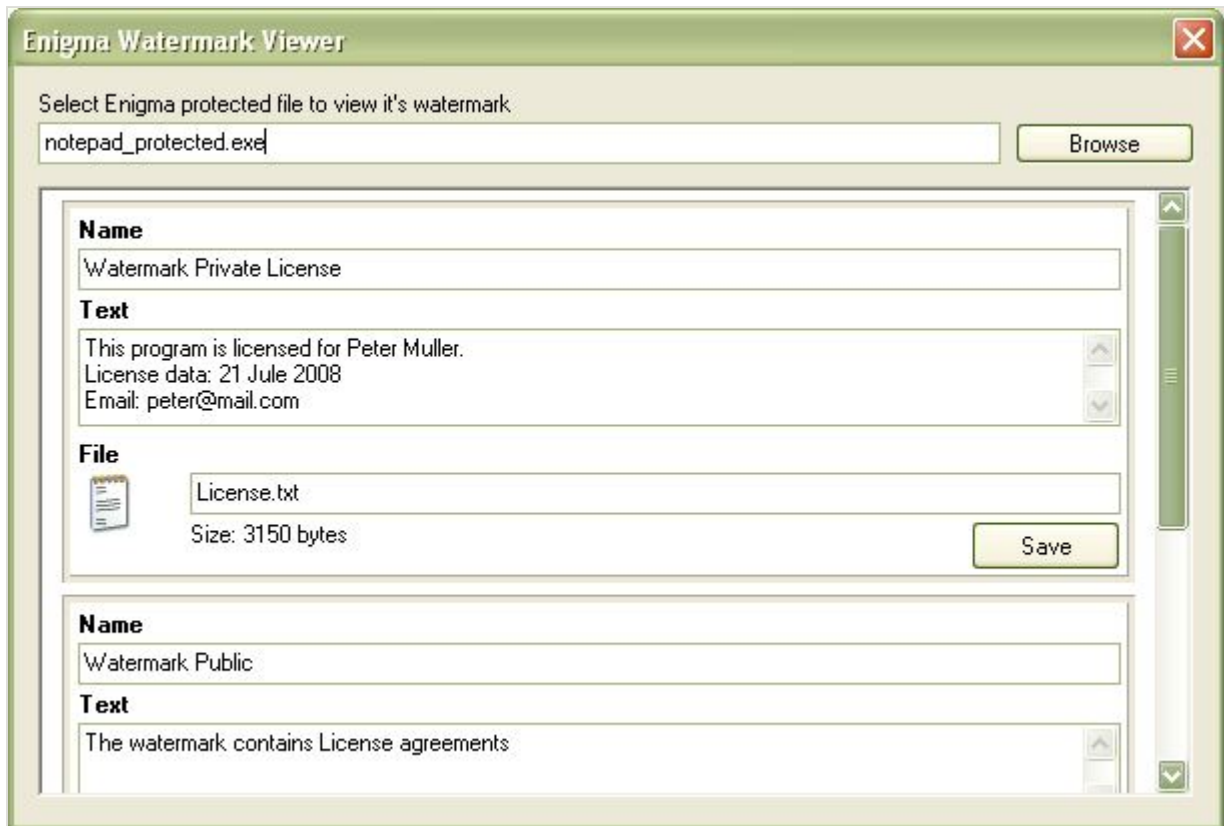
Watermark Name - any string that defines a watermark name.

Watermark Type - the type of a watermark. If the watermark is private, then only the developer who has a valid project file is able to view it. If the type is public, then anyone can view this watermark.

Watermark Text - place any text here.

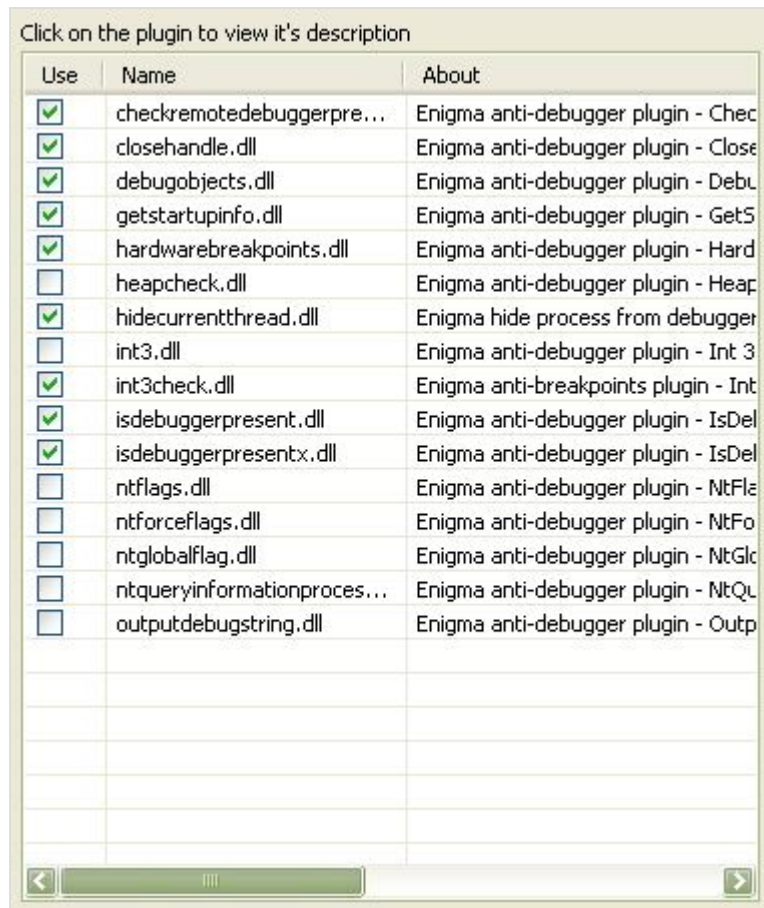
Watermark File - attach any file to the watermark. This file can be extracted with the watermarks viewer.

To view a watermark in the protected file, open the Watermark Viewer, main menu - Tools - Watermark Viewer. Click the Browse button to select the file.



To get the watermarks content at runtime, use [EP_MiscGetWatermark](#) Enigma API.

Plugins



the Enigma Protector supports plugin modules that will be embedded into the protected file and executed when the protected file starts. Plugins for the Enigma Protector are stored in the Plugin subfolder of the installation folder. If you want to add your own plugin, copy it into (for example) c:\program files\the enigma protector\plugins\ folder and open (re-open) the Enigma Protector. Using the plugins system, you may improve the protection, build your own protection systems, enlarge the functionality. It gives you complete control over the protected file together with the Enigma Protector! To embed a plugin into the protected file, check the necessary plugin in the list. To view the plugin description, double-click the plugin in the list.

Plugins SDK

Plugin is simply a DLL (Dynamic Link Library) that exports some functions. The plugin can be compiled with any compiler (MS C++, Borland C++, MASM, Delphi, etc) that supports unmanaged code (meaning it does not support DLLs that were compiled using the .NET technology).

Functions that may export plugin DLLs:

Enigma_Plugin_About - returns a wide char (UNICODE) string that determines the plugin name and author (this info will be shown in the list of plugins, the About column)

Enigma_Plugin_Description - returns a wide char (UNICODE) string that determines the description of a plugin, its functionality (this info will be shown once you double-click the plugin in the list)

Enigma_Plugin_OnInit - a procedure that will be called when the protected module is started. Note that this procedure is called before the protected file is prepared for the execution in the memory. It is the best way to place, for example, your own anti-debugger checkups here.

Enigma_Plugin_OnFinal - a procedure will be called after the final initialization of the protected module in the memory, but before it is executed.

Enigma_Plugin_OnSaveKey - a function is called when the protected file tries to save registration information. It can be used to skip the standard saving routine, to allow more flexibility upon saving registration information. The function has 4 parameters, ARegName and ARegKey contain pointers to registration name and key buffers and ARegNameLen and ARegKeyLen contain the number of bytes of name and key buffers. Note that to handle Wide Strings and UNICODE registration scheme you should divide ARegNameLen and ARegKeyLen values into 2 to get

actual length of the string. This function should return FALSE if it does not save registration information (or it does not need to save it) and TRUE if the function succeeds. Note that standard saving routine will not be called if this function returns TRUE.

Enigma_Plugin_OnLoadKey - a function is called when the protected file tries to load registration information. It can be used to skip the standard loading routine, to allow more flexibility upon loading registration information. The function has 4 output parameters. After the successful loading, ARegName and ARegKey should contain pointers to the buffers of the registration name and key, and ARegNameLen and ARegKeyLen should contain the number of bytes of name and key buffers. Note that you have to allocate buffers for ARegName and ARegKey manually (somewhere in the global space). To handle the Wide Strings and UNICODE registration scheme, you should multiply the lengths of the strings by 2. This function should return FALSE if it does not load registration information (or it does not need to load it) and TRUE if the function succeeds. Note that a standard loading routine will not be called if this function returns TRUE.

Custom VERSION Resource

Add Custom VERSION Resource

Module Version Number

Major Version	Minor Version	Release	Build
1	0	0	0

Module Attributes

<input type="checkbox"/> Debug Build	<input type="checkbox"/> Special Build	<input type="checkbox"/> DLL
<input type="checkbox"/> Pre-release	<input type="checkbox"/> Private Build	

Language

Locale ID: 0x0409

English (United States)

Key	Value
CompanyName	The Enigma Protector Developers Team
FileSecription	Software Protection Tool
FileVersion	1.0.0.0
InternalName	ENIGMA.EXE
LegalCopyright	Copyrights (C) 2002-2009 Vladimir Suk...
LegalTrademarks	Trademarks (R) 2002-2009 Vladimir Su...
OriginalFilename	enigma.exe
ProductName	The Enigma Protector
ProductVersion	1.0.0.0
Comments	http://enigmaprotector.com/

Allows you to add a custom VERSION resource to the protected file. This feature may be important for the existing files that do not have such resources or if a compiler does not support version adding. Edit the necessary version strings in the table for your own application.

To view version resources of the file, open ther folder with the protected file in Explorer and right-click it, select Properties, then, in the appeared window, go to the Version tab. The example is shown below:



Custom MANIFEST Resource



It adds custom MANIFEST resources to the protected file.

It may help in the following cases:

- Visual Styles Common Controls manifest allows you to enable graphical themes of Windows XP/Vista for the protected file,
- MS Visual Studio 2005/2008 manifest allows you to check whether Visual Studio 2005/2008 runtime libraries are installed,
- Each manifest can be edited manually and may contain information about the product developer, its description,
- Manifest may help to define that the protected file should be run under administrative privileges and so on.
- You may select your own .manifest file that should be embedded into the protected file.

The only limitation there is the developer's imagination!

Markers Analyses

The image shows a software interface with two main panels. The top panel, titled 'Found Markers', contains a table with 11 rows of marker information. Below the table is an 'Analyse' button. The bottom panel, titled 'Properties', contains a section for excluding sections from markers analysis, with a table listing various sections like CODE, DATA, BSS, .idata, and .tls, each with checkboxes and columns for Raw Address, Raw Size, and Virtual Address.

#	Name	Virtual Address
1	check_protection_begin	0x004520DC
2	check_protection_end	0x004520F0
3	decrypt_on_execute_begin	0x00452104
4	decrypt_on_execute_end	0x0045211C
5	unprotected_begin	0x00452134
6	unprotected_end	0x00452148
7	unreg_crypt_begin1	0x0045215C
8	unreg_crypt_end1	0x00452170
9	unreg_crypt_begin15	0x00452184
10	unreg_crypt_end15	0x00452198
11	unreg_crypt_begin8	0x004521AC

Analyse

Properties

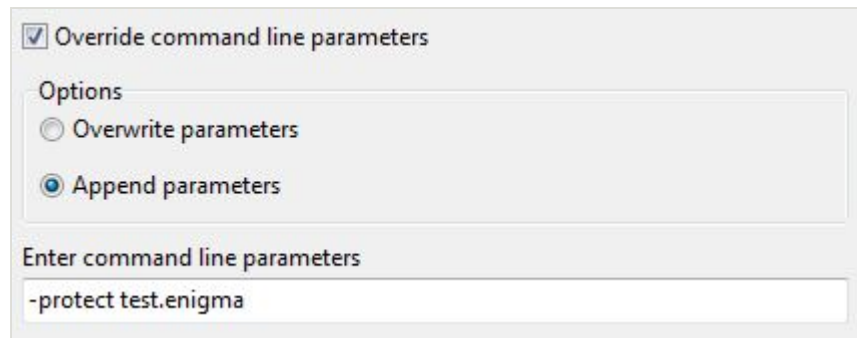
Exclude selected sections from markers analysing:

#	Name	Raw Address	Raw Size	Virtual Address
<input type="checkbox"/>	CODE	0x00000400	0x00051600	0x00001000
<input type="checkbox"/>	DATA	0x00051A00	0x00001200	0x00053000
<input type="checkbox"/>	BSS	0x00052C00	0x00000000	0x00055000
<input type="checkbox"/>	.idata	0x00052C00	0x00002200	0x00056000
<input type="checkbox"/>	.tls	0x00054E00	0x00000000	0x00059000
<input type="checkbox"/>	.data	0x00054E00	0x00002200	0x0005A000

The markers analyses panel can be used as a help tool only, it does not serve any protection purpose. It allows to view markers that are used in the input file. To analyse the file and find all the markers in the code, just click the "Analyse" button. If the list is empty after the analysis, no markers have been found. Read more about [the Markers](#).

All the sections of the input PE file are shown at the bottom of the panel. There, you can exclude certain sections from the markers analysis. What is it needed for? Please remember that this section is used in VERY particular cases. Sometimes, Enigma may fail and find markers in data (or any other) section of the file, it means that some data in your application contain a markers signature. After the wrong markers that are found are protected, the workability of the protected file can be damaged. This panel can be used ONLY in such cases.

Command Line



The image shows a configuration window with a checked checkbox labeled "Override command line parameters". Below this is a section titled "Options" containing two radio buttons: "Overwrite parameters" (unselected) and "Append parameters" (selected). At the bottom, there is a text input field labeled "Enter command line parameters" containing the text "-protect test.enigma".

Allows you to overwrite or append initial command line parameters. The command line parameters entered here will always be passed to the protected module.

Overwrite parameters - any initial command line parameters will be overwritten with the defined ones.

Append parameters - a defined parameters string will be appended to the initial parameters.


```
#include <windows.h>

{
    char variable[256]="";
    if (!GetEnvironmentVariable("my variable", variable, sizeof(variable))) {
        MessageBox(0, "Variable found!", "Application", MB_OK);
    };
}
```

Visual Basic

```
Public Declare Function GetEnvironmentVariable Lib "kernel32.dll" Alias "GetEnvironmentVariableA" (ByVal Name As String, ByVal Buffer As String, ByVal Size As Integer) As String
```

C# (.NET)

```
public class Enigma_EnvironmentVariables
{
    [DllImport("kernel32")] public static extern int GetEnvironmentVariable(string lpName, string lpBuffer, int lpSize);
}
```

Trial Control

This section describes such features as:

- | trial limitations (limitation of the "try" period by a certain criterion);
- | the reminder (it will show a window with a reminder message if the module is unregistered);
- | time control (it controls time reversing).

You can use special Enigma API functions to release your own embedded trial control. Use of Enigma API implies attaching additional modules to yours. These additional modules represent an interface to the Enigma API. Detailed information about the Enigma API can be found in [API description section](#) and in the program examples (the Examples folder in The Enigma Protector installation folder). To disable the trial limitation, one should perform registration of the program module. It can be done using the registration key. The Enigma Protector contains special resources for [registration keys generation](#).

Follow the links below to get more details.

- | [Common](#)
- | [Trial Data Storing](#)
- | [Lock trial to user language](#)
- | [Limitation Of Executions Count](#)
- | [Limitation Of Days Count](#)
- | [Limitation Of Expiration Date](#)
- | [Limitation From Date Till Date](#)
- | [Limitation Of Execution Time](#)
- | [Reminder](#)
- | [Time Control](#)

Common

Reset trial in new version

Trial expiration action

Open file if the trial has expired

File name:
application.exe

Reset trial in new version - resetting the trial in a new version means that the trial limitation will be reset in the new version. If you use limitation by the days count or limitation by the executions count, this function can reset these trial limitations in the new version of the program module. It means that the new version of the program module will count limitation values from the beginning.

Attention: To make this function work correctly, you should define the current version of the module in the [Input](#) section.

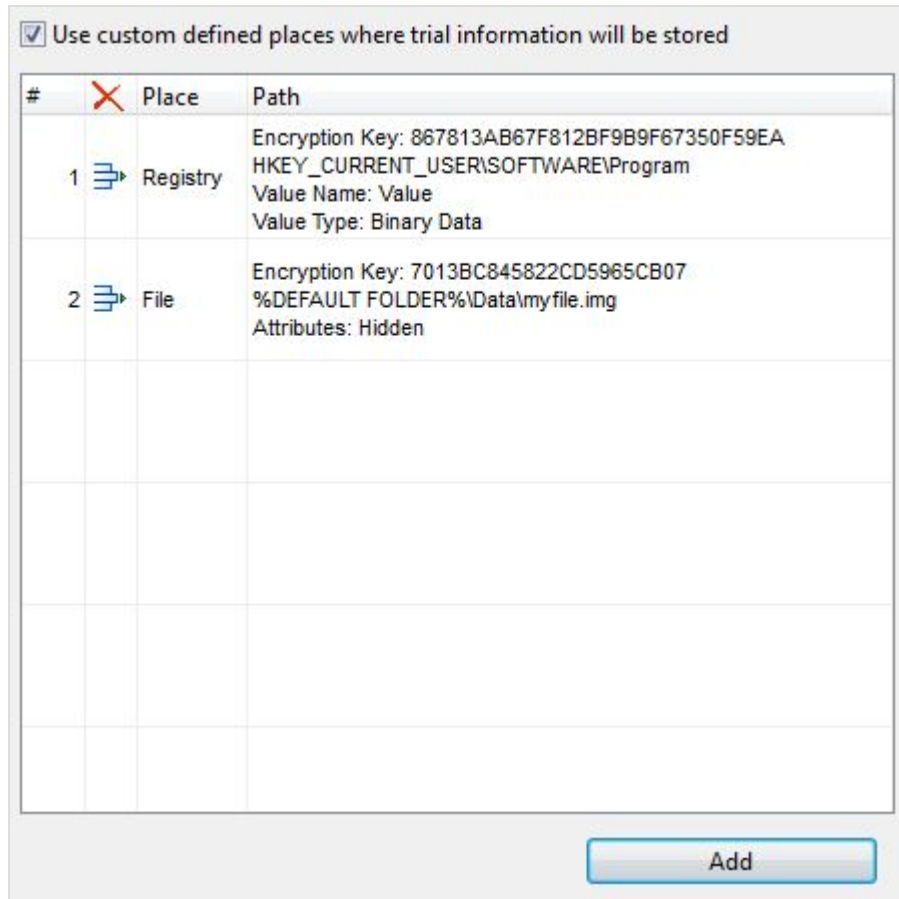
Open file if the trial has expired - allows you to open the file specified in the File Name field when any trial limitation has expired. The file will be opened only if you have chosen the Terminate Execution option for the Trial Expire action. If you have set to keep the file executed upon the trial expiration, the file will not open.

File name - the file name has a wide range of uses, it may be not just a file; hyper and email links can be used as well.

The following examples show you what the File Name field can contain:


The File Name field content	Description
http://www.mysite.com/	hyper link, will open an Explorer window with the same link;
mailto:admin@mysite.com	email link, will open a default email client offering to send an email to the same address;
License.txt	open license.txt file. The file will be opened in a default application which supports .txt format. The file should be placed in the same folder with the protected module;
Relative\Register.exe	open the Register.exe file which is located in the Relative subfolder to the folder of the protected module;
..\Register.exe	open the Register.exe file which is located in the upper folder to the folder of the protected module;
C:\Windows\Register.exe	open the Register.exe file from the c:\Windows absolute folder.

Trial Data Storing

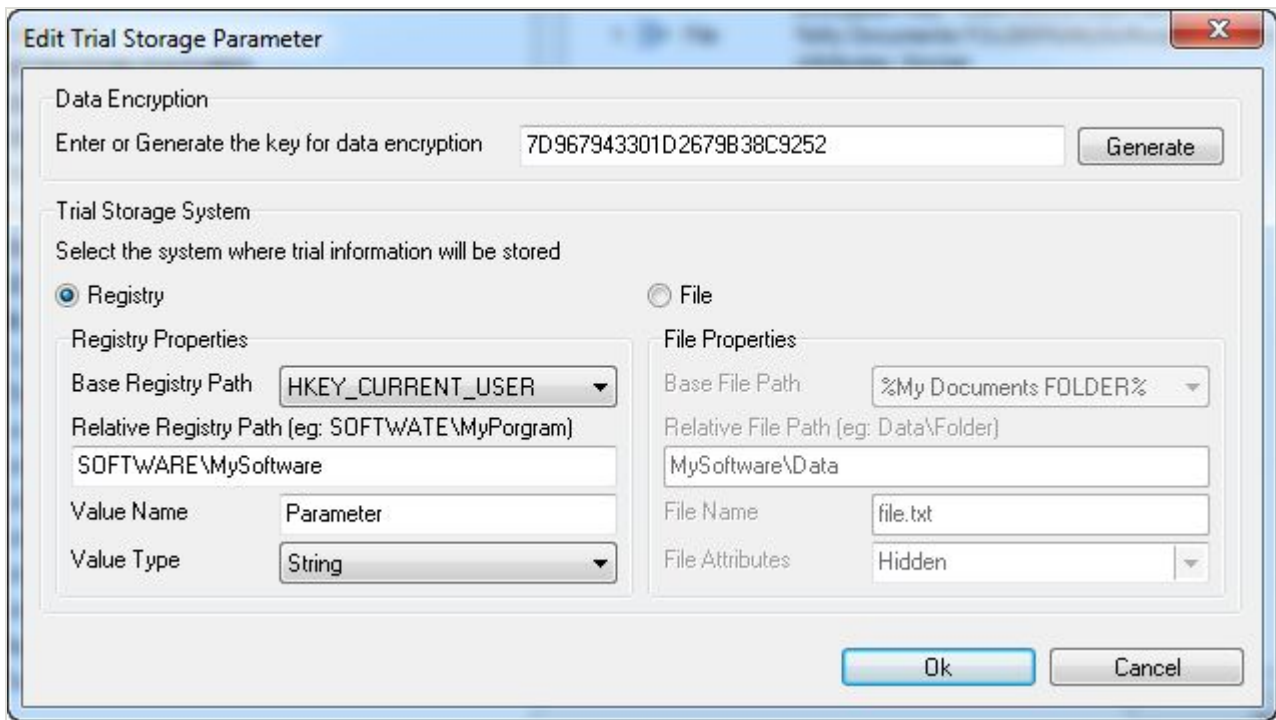


This is a unique feature which allows entering your own paths where the trial information will be stored. What does trial information mean? Trial information is some data including the number of trial executions or days left, etc. For correct processing of the trial parameters the protector has to store the information regarding the expired items somewhere, usually it is a file system or registry. Protection systems do not allow changing trial path parameters - and often the path is well-known to crackers - and trial reset tools that allow resetting the expired trial. This feature will not help avoid trial reset by advanced users, but it is a useful solution to resist automatic tools.

This is not a necessary feature that should always be enabled, it is optional. If you do not use it, the Enigma Protector will work as well and use its own paths to store trial information.

Click the *Add* button to add a path to the list, double click the existing path to modify it, click  to delete an item from the list.

Add/Edit path dialog:



Data Encryption - set the string that will be used for encrypting/decrypting the trial information, this may be any string;

Registry/File - specify the system where the registration info will be stored - in the registry or in a file;

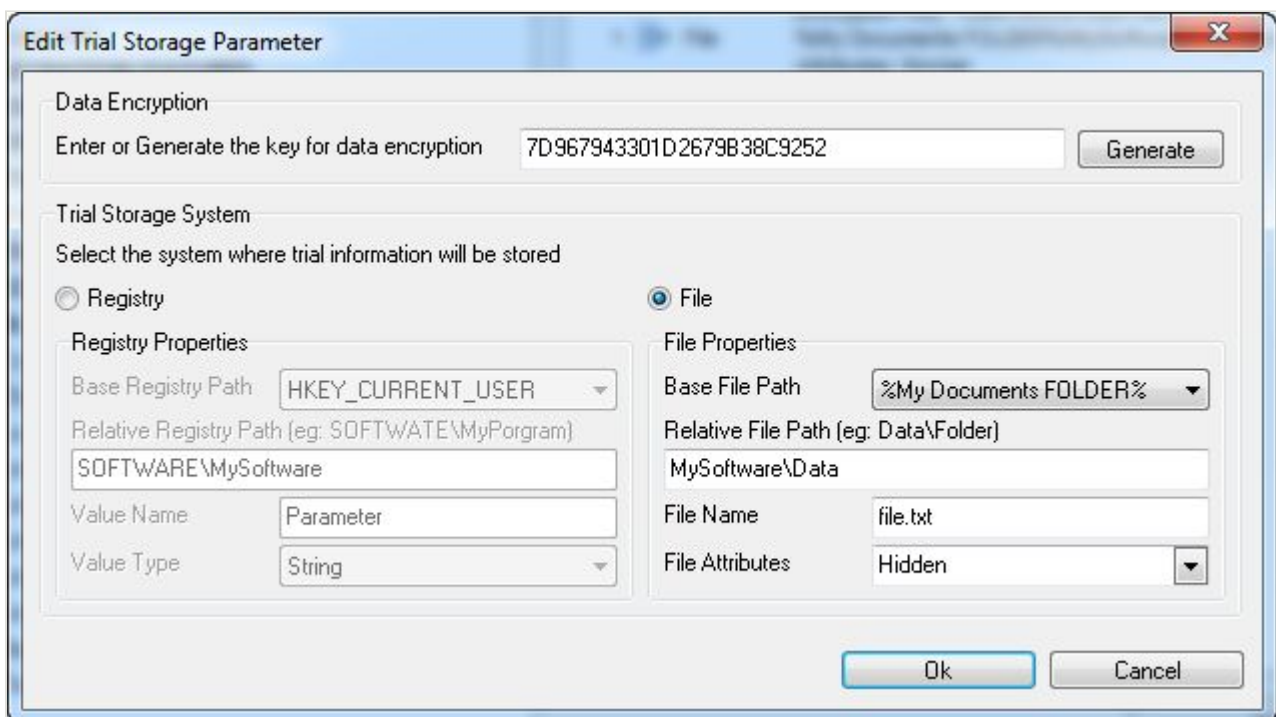
Base Registry Path - a root registry item where the registration information will be stored. Please note:

- 1 we recommend using HKEY_CURRENT_USER as a base, because this branch of registry is specially designed for storing different information of the currently logged user;
- 1 trial information stored in HKEY_LOCAL_MACHINE will be valid for any user account - if the trial has expired in one user account, all other users will have it expired as well;
- 1 HKEY_CLASSES_ROOT, HKEY_LOCAL_MACHINE, HKEY_USERS, HKEY_CURRENT_CONFIG can be disabled for writing for non-admin user accounts and the protection may not work properly.

Relative Registry Path - a relative path in the registry to store registration information;

Value Name - a registry value name;

Value Type - the type of the registry value, can be String or Binary data;



Base File Path - the base file path;

- | if using folders %My Documents FOLDER%, %Program Files FOLDER%, %Program Files\Common FOLDER%, %AllUsers\Documents FOLDER%, %AllUsers\ApplicationData FOLDER%, %My Pictures FOLDER%, %History FOLDER%, %Cookies FOLDER%, %InternetCache FOLDER% the application will require Internet Explorer for Windows 95, Windows 98, Windows NT 4.0. If older version of Internet Explorer are installed, the folder will be changed to %DEFAULT FOLDER%
- | if using folders %My Documents FOLDER%, %My Pictures FOLDER%, %History FOLDER%, %Cookies FOLDER%, %InternetCache FOLDER% the trial information will be stored for the current user account, otherwise the trial will be applied to all Windows users.

Relative File Path - a relative file path, this feature is used for adding a Base File Path;

File Name - the name of the file where registration information will be stored;

File Attributes - attributes of the file, can be a combination of Read Only, Archive, System and Hidden attributes. If nothing is entered, normal attributes are applied to the file.

Limitation By Executions Count

The screenshot shows a configuration window with the following elements:

- A checked checkbox labeled "Use limitation of executions count".
- A label "Maximal count of executions:" followed by a numeric input field containing "100".
- A section titled "Action when expired" containing:
 - A checked checkbox labeled "Terminate execution after (seconds)" followed by a numeric input field containing "0".
 - A checked checkbox labeled "Show message before termination" followed by a "Design message" button.

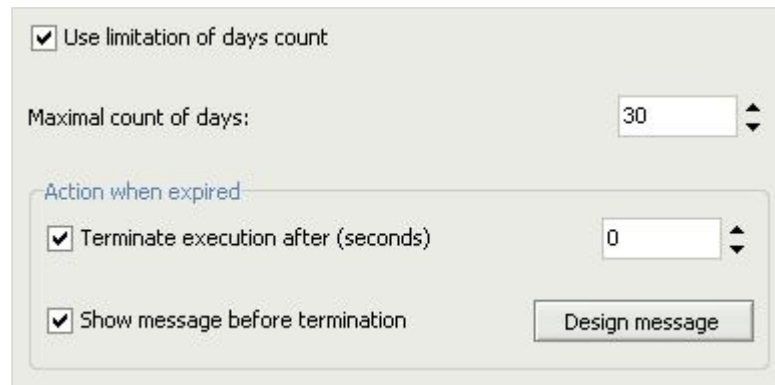
Use limitation by executions count - check this box to enable the respective function. The main idea of this function is as follows: on the first execution of the protected module a special record is created in the Windows registry. This record contains the number of times the program module is executed. When the protected module starts up, The Enigma Protector loader checks the current count of executions. If it is greater than the maximum limit (defined in "Maximal count of executions" field before protection), the execution of the program module is stopped.

You can check the limitation parameters during program module execution with the help of [Enigma API](#). Use the [EP_TrialExecutions](#) function to get the total and remaining number of executions. To reset the trial period you can use the functions of registration key verification. If The Enigma Protector loader detects a correct valid registration key in the system, the trial period will be reset.

Terminate execution after (seconds) - if the trial period is expired, the execution of the program module will be stopped after X seconds. If this parameter is set equal to zero, the execution of the module stops before the deciphering and executing of the main code of the module.

Show Message before termination - check this box if it is necessary to inform the user that the trial period is expired and the execution of the module will be stopped. You can edit the content of the message by clicking the "[Design Message](#)" button. If this function is disabled, the execution of the program module will be stopped without any notifications.

Limitation by Days Count



The screenshot shows a configuration dialog box with the following elements:

- A checked checkbox labeled "Use limitation of days count".
- A label "Maximal count of days:" followed by a numeric spinner box containing the value "30".
- A section titled "Action when expired" containing:
 - A checked checkbox labeled "Terminate execution after (seconds)" followed by a numeric spinner box containing the value "0".
 - A checked checkbox labeled "Show message before termination" followed by a button labeled "Design message".

Use limitation by days count - check this box to enable the respective function. The main idea of this function is to limit the trial period by count of days the protected application will work since the first execution. The maximum number of days must be defined in the respective field before protection of the program module.

On the first execution of the protected module a special record is created in the Windows registry. This record contains the number of days the protected module operates. When the protected module is started, The Enigma Protector loader checks this value. If it is greater than the maximum limit, the execution of the program module is stopped.

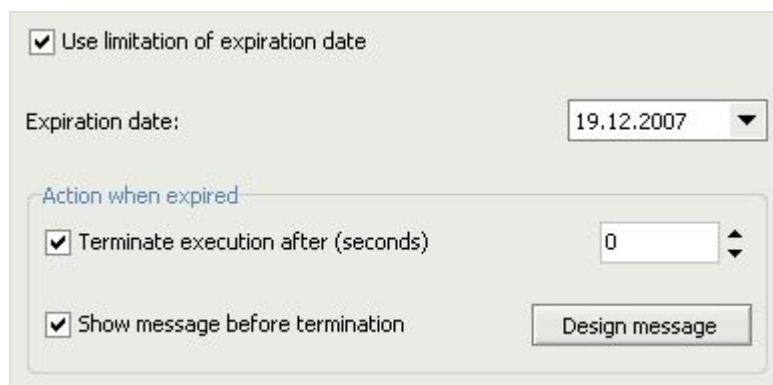
You can check limitation parameters during program module execution with the help of [Enigma API](#). Use the [EP_TrialDays](#) function to get the total and remaining count of executions. To reset the trial period you can use the functions of registration key verification. If The Enigma Protector loader detects a valid registration key in the system, the trial period will be reset.

Note: To prevent setting the system clock backward use the [TrialControl->TimeControl](#) function.

Terminate execution after (seconds) - if the trial period is expired, the execution of the program module will be stopped after X seconds. If this parameter is set equal to zero, the execution of the module stops before the deciphering and executing of the main code of the module.

Show Message before termination - check this box if it is necessary to inform the user that the trial period is expired and the execution of the module will be stopped. You can edit the content of the message by clicking the "[Design Message](#)" button. If this function is disabled, the execution of the program module will be stopped without any notifications.

Limitation By Expiration Date



The screenshot shows a configuration dialog box with the following elements:

- A checked checkbox labeled "Use limitation of expiration date".
- An "Expiration date:" label followed by a date picker showing "19.12.2007".
- A section titled "Action when expired" containing:
 - A checked checkbox labeled "Terminate execution after (seconds)" followed by a numeric spinner set to "0".
 - A checked checkbox labeled "Show message before termination" followed by a "Design message" button.

Use limitation by expiration date - check this box to enable the respective function. The main idea of this function is as follows: when the protected module starts up, The Enigma Protector loader checks the current date and compares it to the expiration date (date when the "try" period is expired). If the trial period is expired, the execution of the program module is stopped.

You can check the limitation parameters during program module execution with the help of [Enigma API](#). Use the [EP_TrialExecutionDate](#) function to get the year, month and day of the expiration date. To reset the trial period you can use the functions of registration key verification. If The Enigma Protector loader detects a valid registration key in the system, the trial period will be reset.

Note: To prevent setting the system clock backward use the [TrialControl->TimeControl](#) function.

Terminate execution after (seconds) - if the trial period is expired, the execution of the program module will be stopped after X seconds. If this parameter is set equal to zero, the execution of the module stops before the deciphering and executing of the main code of the module.

Show Message before termination - check this box if it is necessary to inform the user that the trial period is expired and the execution of the module will be stopped. You can edit the content of the message by clicking the "[Design Message](#)" button. If this function is disabled, the execution of the program module will be stopped without any notifications.

Limitation From Date Till Date

The image shows a configuration dialog box titled "Limitation from date till date". It contains the following elements:

- A checked checkbox labeled "Use limitation from date to date".
- A "Start trial date:" label followed by a date picker showing "1/ 1/2008".
- An "End trial date:" label followed by a date picker showing "1/ 1/2009".
- A section titled "Action when expired" containing:
 - A checked checkbox labeled "Terminate execution after (seconds)" followed by a numeric input field containing "0".
 - A checked checkbox labeled "Show message before termination" followed by a "Design message" button.

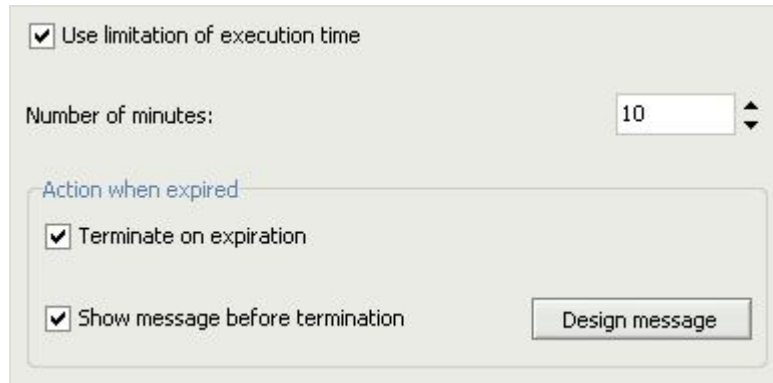
Use limitation from date till date - the function enables trial limitation from one date till another. Once a protected module is executed, Enigma loader reads the current date, and if the current date precedes the *Start trial date* or follows the *End trial date*, the trial period is deemed expired.

You can check the limitation parameters during program module execution with the help of [Enigma API](#). Use the [EP_TrialDateTillDate](#) function to get the trial start and end dates. To reset the trial period you can use the functions of registration key verification. If The Enigma Protector loader detects a valid registration key in the system, the trial period will be reset.

Terminate execution after (seconds) - if trial period is expired, the execution of the program module will be stopped after X seconds. If this parameter is set equal to zero, the execution of the module stops before deciphering and executing the main code of the module.

Show Message before termination - check this box if it is necessary to inform the user that the trial period is expired and the execution of the module will be stopped. You can edit the content of the message by clicking the "[Design Message](#)" button. If this function is disabled, the execution of the program module will be stopped without any notifications.

Limitation Of Execution Time



The image shows a configuration dialog box titled "Limitation Of Execution Time". It contains the following elements:

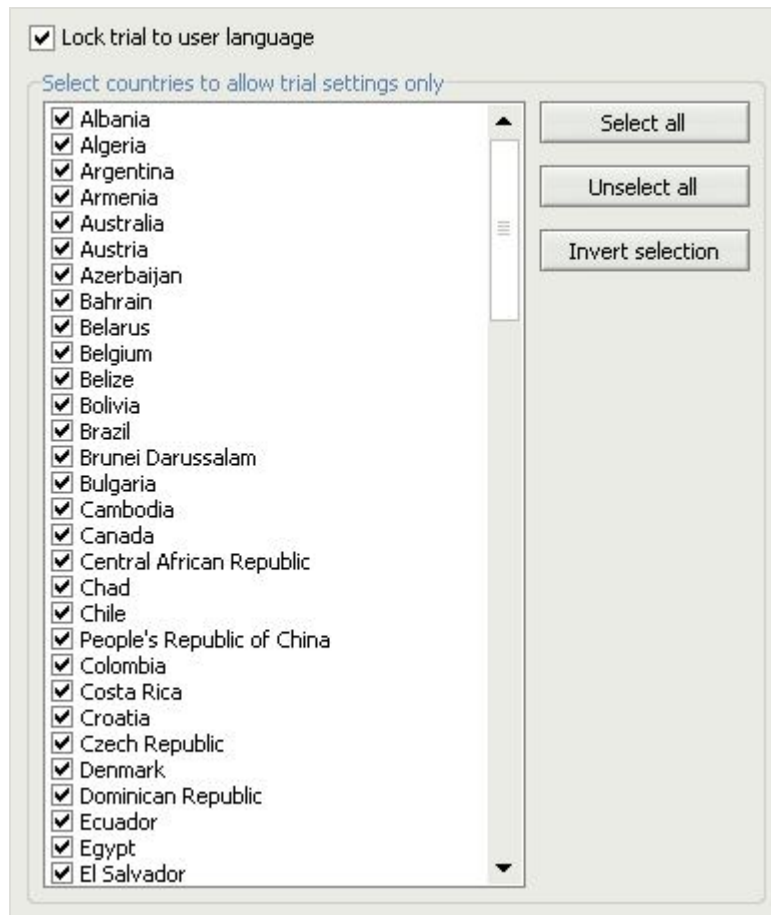
- A checked checkbox labeled "Use limitation of execution time".
- A label "Number of minutes:" followed by a spin box containing the value "10".
- A section titled "Action when expired" containing:
 - A checked checkbox labeled "Terminate on expiration".
 - A checked checkbox labeled "Show message before termination".
 - A button labeled "Design message".

Use limitation of execution time - use this function to limit the trial execution time. If this option is enabled, the Enigma loader counts the number of minutes since the module start and if this number exceeds the value defined in the "Number of minutes" field, the trial expiration action is triggered. When the minute count reaches the defined value, the trial will be deemed expired each time the protected module is executed. To remove the trial limitation, the module needs to be registered. Once the module is registered, the trial limitation will be lifted and the user will be able to work with the module for any time per execution. If you need to check the total and remaining number of minutes during module operation, you can use the [Enigma API](#). The [EP_TrialExecutionTime](#) function returns the total number of trial minutes and the number of minutes remaining counted since the module start. To reset the trial period you can use functions of registration key checking. If The Enigma Protector loader detects a valid registration key in the system, the trial period will be reset.

Terminate on expiration - if the trial period is expired, the execution of the program module will be stopped. If this parameter is set equal to zero, the execution of the module stops before the deciphering and executing of the main code of the module.

Show Message before termination - check this box if it is necessary to inform the user that the trial period is expired and the execution of the module will be stopped. You can edit the content of the message by clicking the "[Design Message](#)" button. If this function is disabled, the execution of the program module will be stopped without any notifications.

Lock Trial To User Language



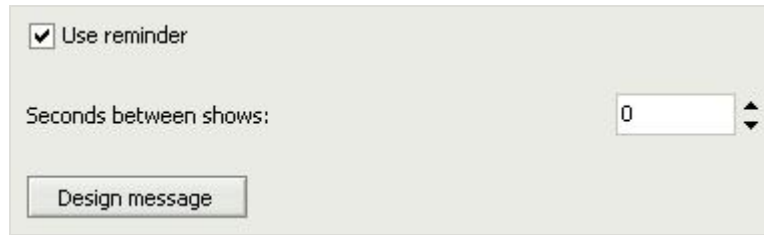
Lock trial to user language - this function locks trial settings to the specified user country. Select the countries where the trial setting will be effective in the countries list. If the protected module is executed in a country which has not been selected, all trial limitations will be lifted, i.e. the execution of the module won't be limited. This option affects all trial limitations set in the TRIAL CONTROL panel;

Select all - select all items in the countries list;

Deselect all - deselect all items in the countries list;

Invert selection - invert selection of all items, i.e. deselect all selected items and vice versa.

Reminder



Use reminder

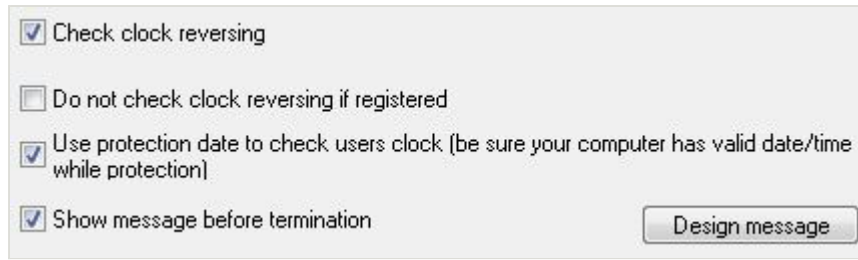
Seconds between shows:

Design message

Use reminder - check this box to enable the reminder feature. If this feature is enabled, a reminder message window will be shown when the protected module starts up. The contents of the message (for example, a notification of necessary registration of the module) can be defined after clicking the "[Design Message](#)" button. To disable the reminder, the user must register the program module. If The Enigma Protector loader detects a valid registration key in the system, this message will not be shown.

Seconds between shows - this value defines the period of time after which the reminder message will be shown again. If this value is set to zero, the reminder message will be shown only once when the program module starts up.

Time Control



Check clock reversing

Do not check clock reversing if registered

Use protection date to check users clock (be sure your computer has valid date/time while protection)

Show message before termination

Design message

Check clock reversing - check this box to use the system clock reverse control function. It helps prevent setting the system clock backward by the user for evasion of trial limitations or limitations of date-dependent keys. When the protected module starts up, The Enigma Protector loader checks if system time has been changed. If it has, the loader will stop module execution.

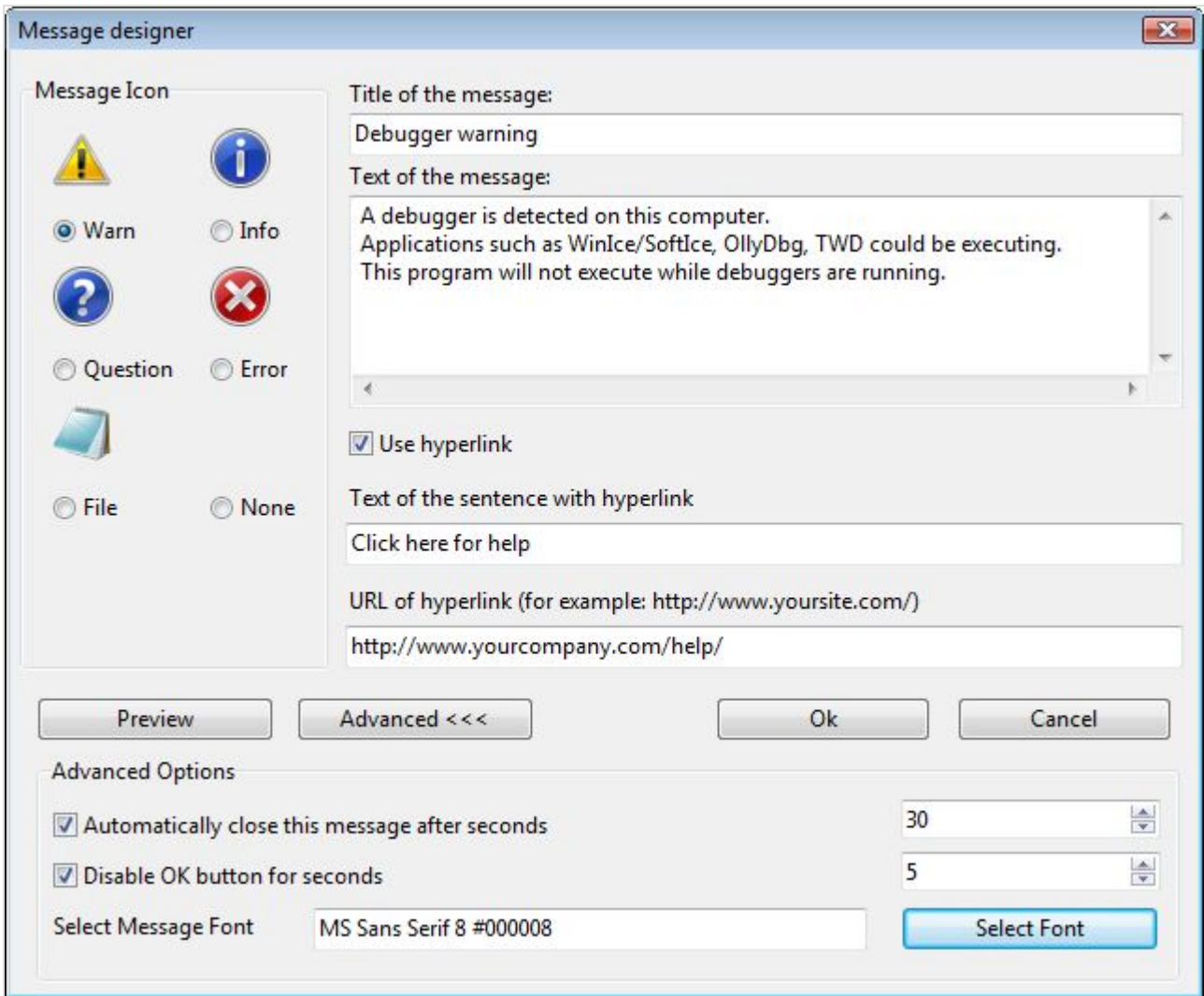
Do not check clock reversing if registered - use this function to disable user time verification if the application is registered. Please note, it is not recommended to disable clock reversing check if registration keys with (eg) expiration date are being used.

Use protection date to check users clock - if the option is enabled, the Enigma Protector will also check if the current date on the user PC is not earlier than the protection date. If the user PC date is earlier than the date of protection, Enigma will deem the clock set backward. Please note, you have to be sure your system clock (and Time Zone information) is correct.

Attention: Presence of a valid registration key does not affect the function.

Show Message before termination - check this box if you want to inform the user that the system clock has been reversed, and to offer restoring the system clock for correct execution of the protected module. You can design the contents of the message by clicking the ["Design Message"](#) button. If this function is disabled, the execution of the program module will be stopped without any notifications.

Message Designer



Message designer is a dialog window for editing messages for different purposes.

Use hyperlink - check this box if you want to place a sentence with the hyperlink on the message window.

Text of the sentence with hyperlink - these words contain semantics of the link.

URL of hyperlink - this string will be forwarded to the user's web browser.

In the trial messages you can insert special strings which will be replaced by the corresponding values:

Variable string	Description	Related Link
%AppName%	Application Name variable from the project inputs	Input
%AppVers%	Application Version variable from the project inputs	Input
%CU_EXTFILES%	Name of the external file that has failed with the external files checkup	Checkup of External Files
%CU_EXECP%	File name of the executed process that has failed with the executed processes checkup	Checkup of Executed Processes
%CU_INSTSERV%	Name of the service that has failed with the installed services checkup	Checkup of Installed Services
%CU_WINVER%	Full version of the current Windows	Checkup of Windows Version
%CU_VIRTTOOLS%	Name of the Virtual Machine	Checkup of Virtualization Tools
%HardwareID%	Current Hardware ID	Hardware Lock
%RegName%	Registration Name	Keys generator
%RegKey%	Registration Key	
%KeyExpYear%	Year of key expiration date	
%KeyExpMonth%	Month of key expiration date	
%KeyExpDay%	Day of key expiration date	
%TrialExecsTotal%	Total number of trial executions	Limitation of Executions

%TrialExecsLeft%	Remaining number of trial executions	Count
%TrialDaysTotal%	Total number of trial days	Limitation of Days Count
%TrialDaysLeft%	Remaining number of trial days	
%TrialExpYear%	Year of trial expiration date	Limitation of Expiration Date
%TrialExpMonth%	Month of trial expiration date	
%TrialExpDay%	Day of trial expiration date	
%TrialStartYear%	Year of start trial date	Limitation from Date till Date
%TrialStartMonth%	Month of start trial date	
%TrialStartDay%	Day of start trial date	
%TrialEndYear%	Year of end trial date	
%TrialEndMonth%	Month of end trial date	
%TrialEndDay%	Day of end trial date	
%TrialExecMinsTotal%	Total number of trial execution minutes	Limitation of Execution Time
%TrialExecMinsLeft%	Remaining number of trial execution minutes	

The variable strings are non case-sensitive.

Message Icon - select the icon that will appear on the message. There are few kinds of icons: Warn (warning), Info (information), Question, Error, File (associated icon of the protected file), None (without icon).

Automatically close this message after seconds - if the option is enabled then this message will be closed within a defined number of seconds after the message has been shown.

Disable OK button for seconds - it disables OK button for the defined number of seconds, within this time it will be impossible to close message window.

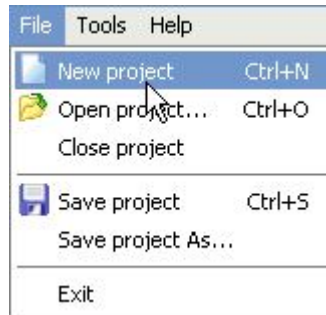
Select Message Font - select the font that will be assigned to the whole message window.

Preview - click this button to take a look at how the message window will look.

Basic operations

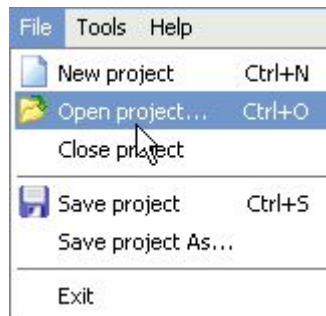
Creation of a new project

To create a new project file select the *"New project"* item. See [Creation of a protection project](#).



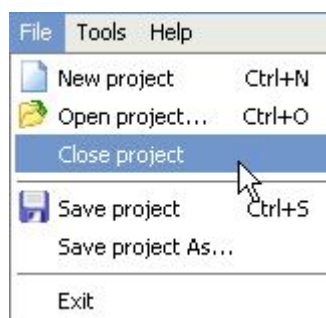
Opening the existing project

To open the existing project file, select the *"Open project..."* item.



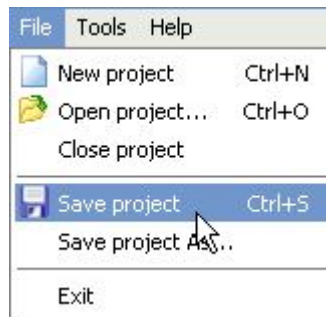
Closing the project

To close the project opened, select the *"Close project"* item.



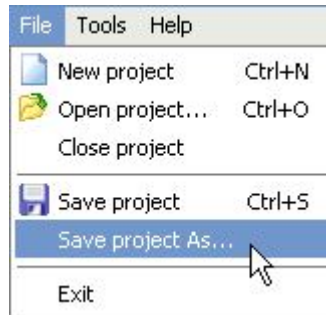
Saving the project

To save the project, select the *"Save project"* item.



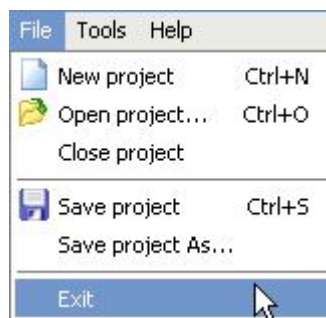
Saving the project with a new file name

To save the project with a new file name, select the "Save project As..." item.



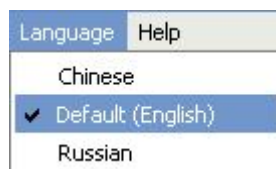
Exiting The Enigma Protector

To close The Enigma Protector, select the "Exit" item.



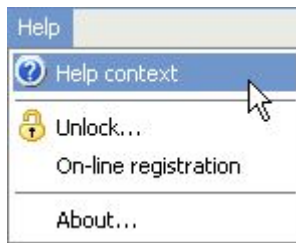
Changing the currently installed GUI language

To change GUI language select necessary language in the "Language" item. The language files should be placed into \lang subfolder with the installed Enigma Protector.



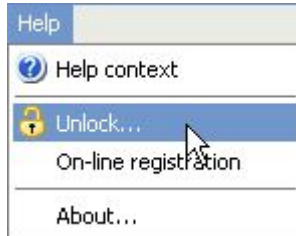
Calling the manual

"Help context" allows to open The Enigma Protector manual.



Unlocking/Registration

Allows you to perform unlocking/registration of The Enigma Protector. See [Order](#) page.



On-line registration

On-line registration provides you with necessary information about purchasing The Enigma Protector. You will be redirected to The Enigma Protector website. After the registration you will have a valid pair of the registration name and registration key by means of which you will be able to unlock registered features of The Enigma Protector.



Calling the registration keys generator

For additional information about using The Enigma Protector registration keys generator see [Keys Generator](#)



About information

The Enigma Protector about dialog.



Reset local trial information

This feature is used for debugging the opened project. In development, resetting local trial information is sometimes needed for playing with The Enigma Protector trial parameters, use this for clearing trial information on the local PC.



Reset local registration information

This feature is used for debugging the opened project. In development, resetting local trial information is sometimes needed for playing with The Enigma Protector registration features, use this for clearing registration information on the local PC.



Reset local special information

The feature allows to reset/delete local special information. It is used for debugging purposes on the developer's side only. The special local information means, for example, [Startup Password](#) information.

Choosing the project parameters:

Choosing the project parameters is performed by means of a graphic user interface. All the items are systematized in tree categories. For each category there is its own page where management elements are placed.

- SETTINGS**
 - Project details
 - Input
 - Output
 - Compression
- REGISTRATION FEATURES**
 - Common
 - Registration data storing
 - Hardware lock
- CHECK-UP**
 - Anti-debugger
 - Control sum
 - Startup password
 - File name
 - Disk drive
 - Executed copies
 - User language
- PROTECTION FEATURES**
 - File analyzer deception
 - Original file size preservation
 - Extra resource protection
 - Advance force import protection
 - WinAPI redirection
 - WinAPI emulation
 - Protected DLL attachment
- OBFUSSION**
 - File entry point
 - Virtual machine
- MISCELLANEOUS**
 - Splash screen
- TRIAL CONTROL**
 - Common
 - Lock trial to user language
 - Limitation of executions count
 - Limitation of days count
 - Limitation of expiration date
 - Limitation of execution time
 - Reminder
 - Time control

Creating a map file

This topic explains how map files can be created for applying in Virtual Machine. Current version of The Enigma Pro supports map files created in Delphi, Borland C++ Builder and MS Visual Studio.

[Delphi up to version 7](#)

[Delphi 2007 and later \(2009, 2010\)](#)

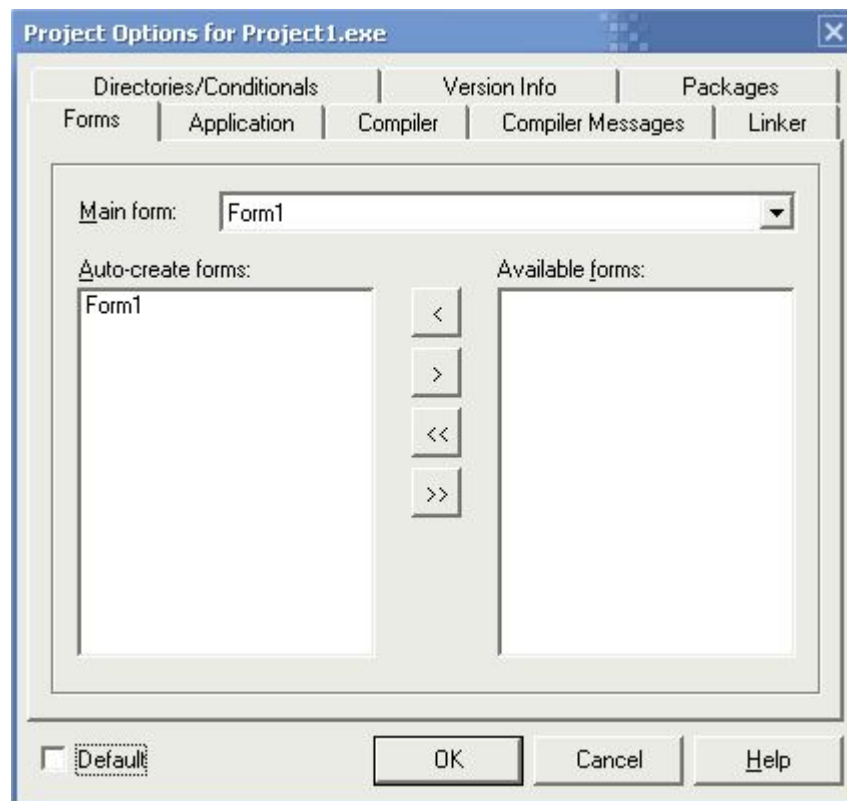
[Borland C++ Builder up to version 7](#)

[Microsoft Visual C++ up to 6.0](#)

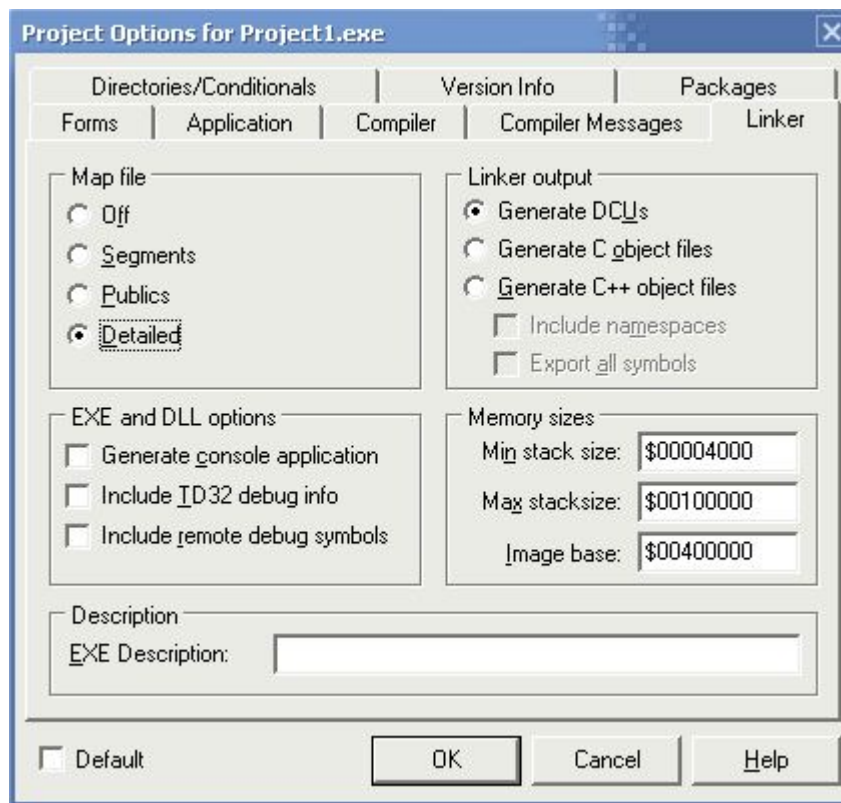
[Microsoft Visual Studio 2003 and later \(2005, 2008\)](#)

Creating a map file in Delphi:

- Open the project of your module in Delphi.
- In the Main Menu select "Project" -> "Options", or just press Shift+Ctrl+F11. You will see the following window:



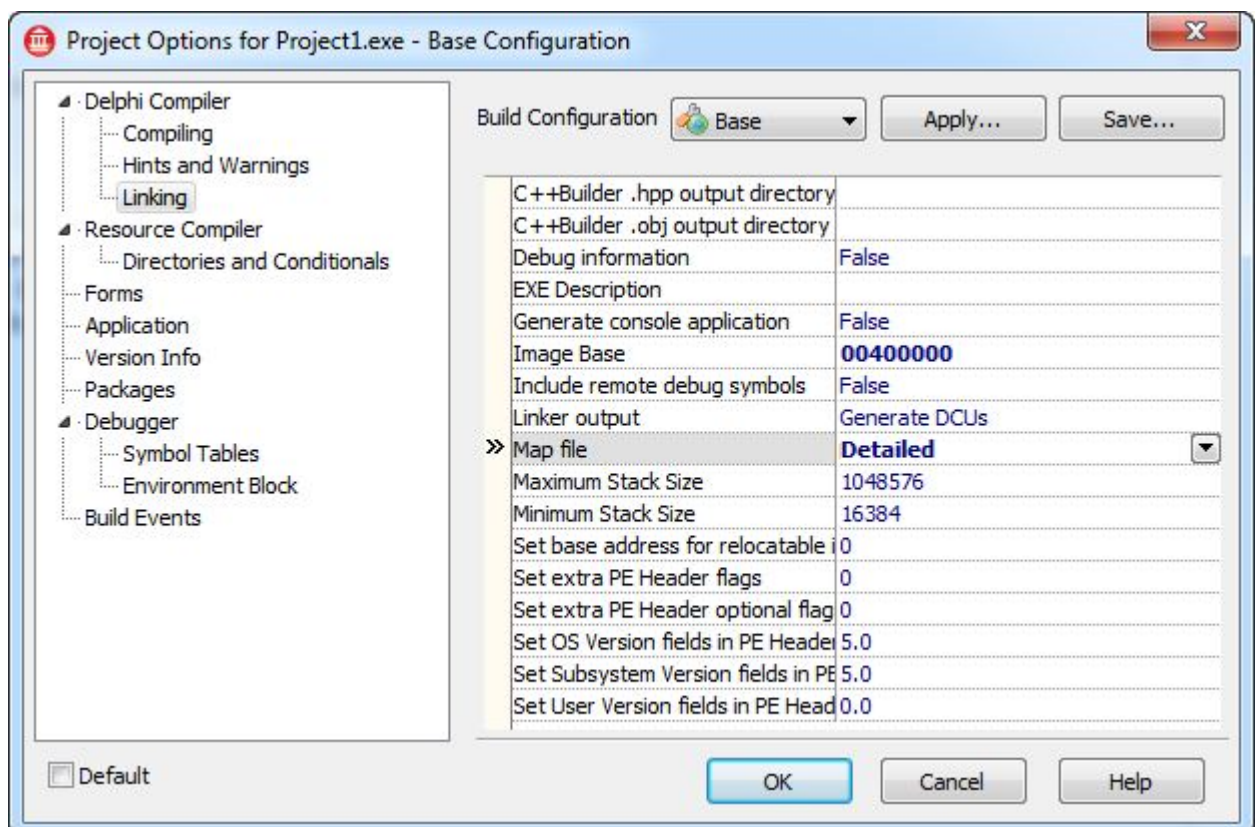
- Select the "Linker" tab and select the "Detailed" radio button on the "Map File" panel:



- Compile your project. In the output folder you should see the executable file of your module and the same file with the .map extension. This .map file can be used to enable "Virtual Machine" features.

Creating a map file in Delphi 2007 and later:

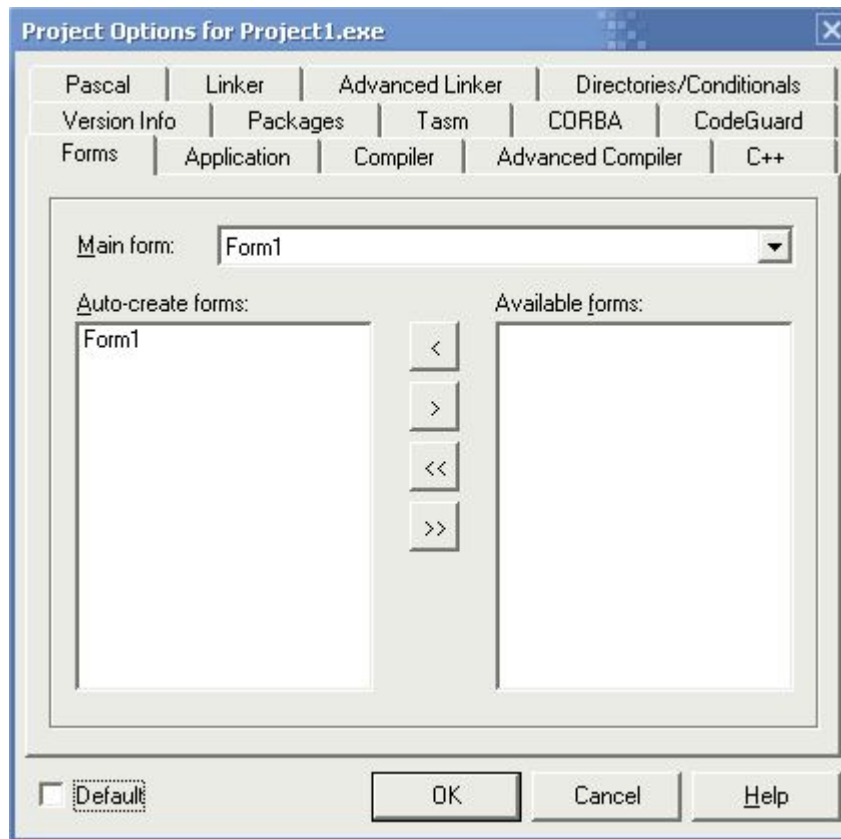
- Open the project of your module in Delphi;
- In the Main Menu select "Project" -> "Options", or just press Shift+Ctrl+F11. In the appeared window select "Linker" item in the left navigation panel, then select "Map file - Detailed";



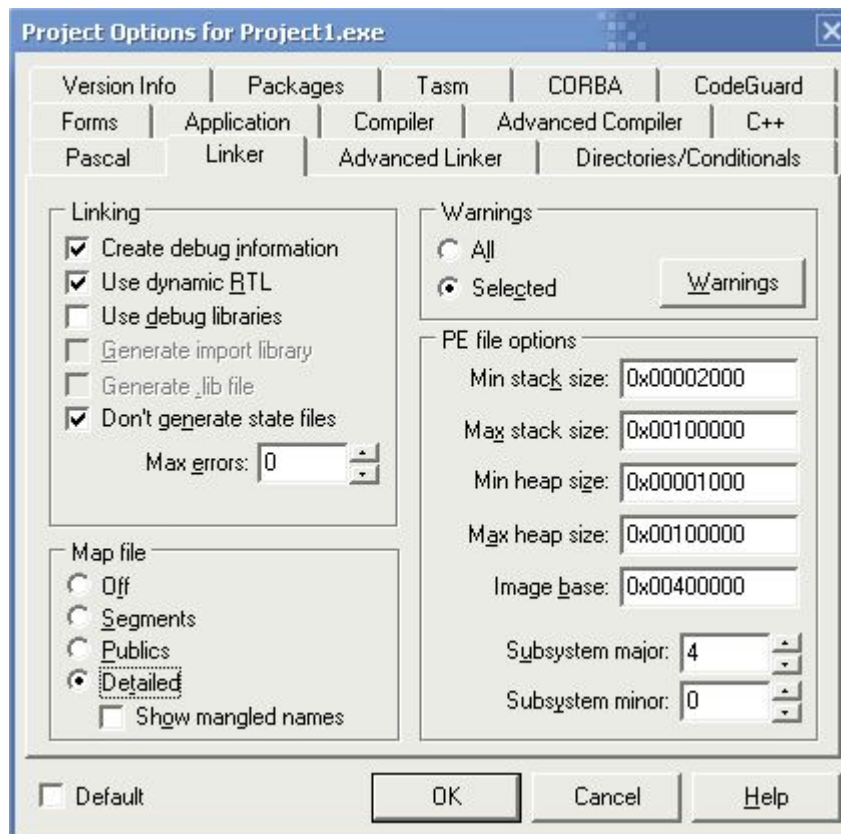
- Compile your project. In the output folder you should see the executable file of your module and the same file with the .map extension. This .map file can be used to enable "Virtual Machine" features

Creating a map file in Borland C++ Builder:

- Open the project of your module in C++ Builder



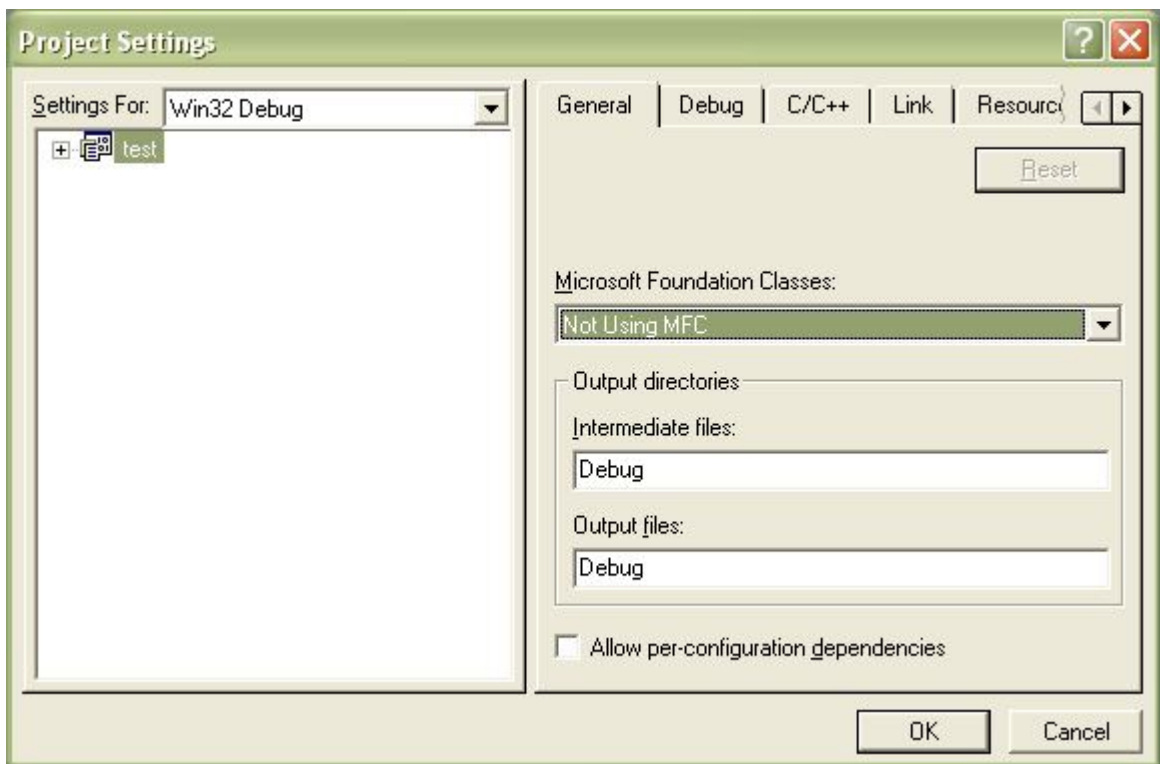
- Select the "Linker" tab and select the "Detailed" radio button on the "Map File" panel:



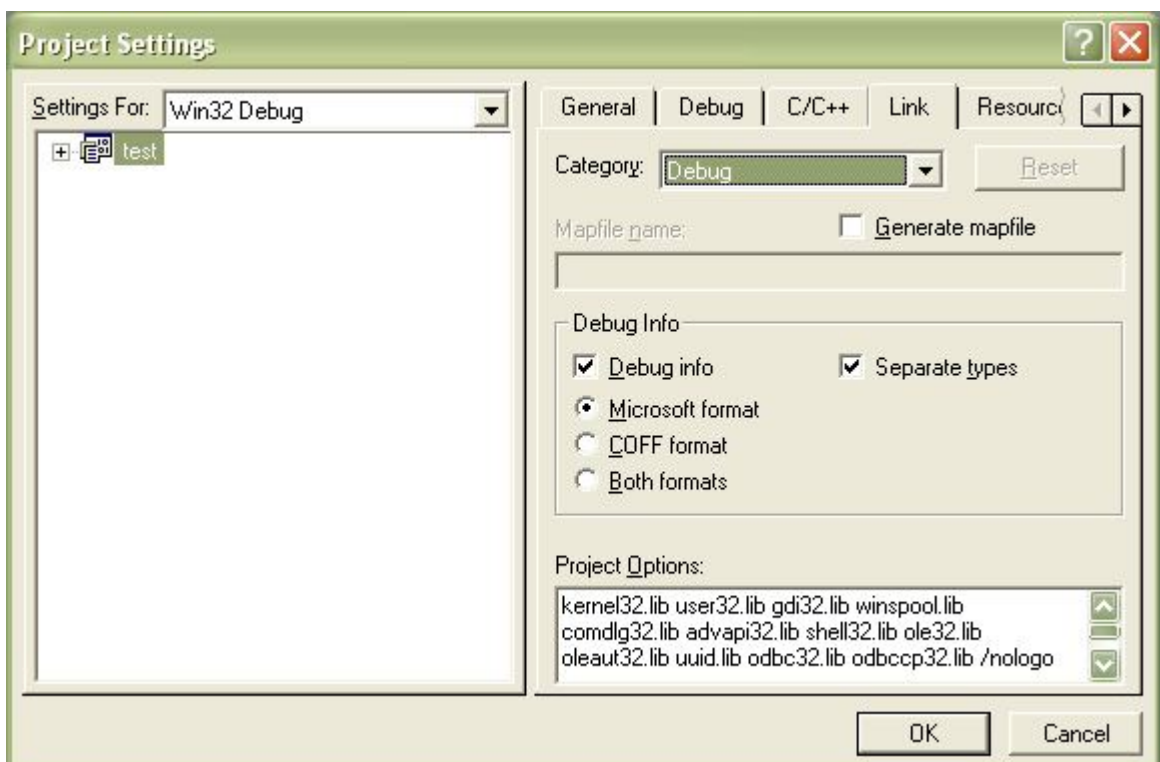
- Compile your project. In the output folder you should see the executable file of your module and the same file with the .map extension. This .map file can be used to enable "Virtual Machine" features.

Creating a map file in MS Visual C++:

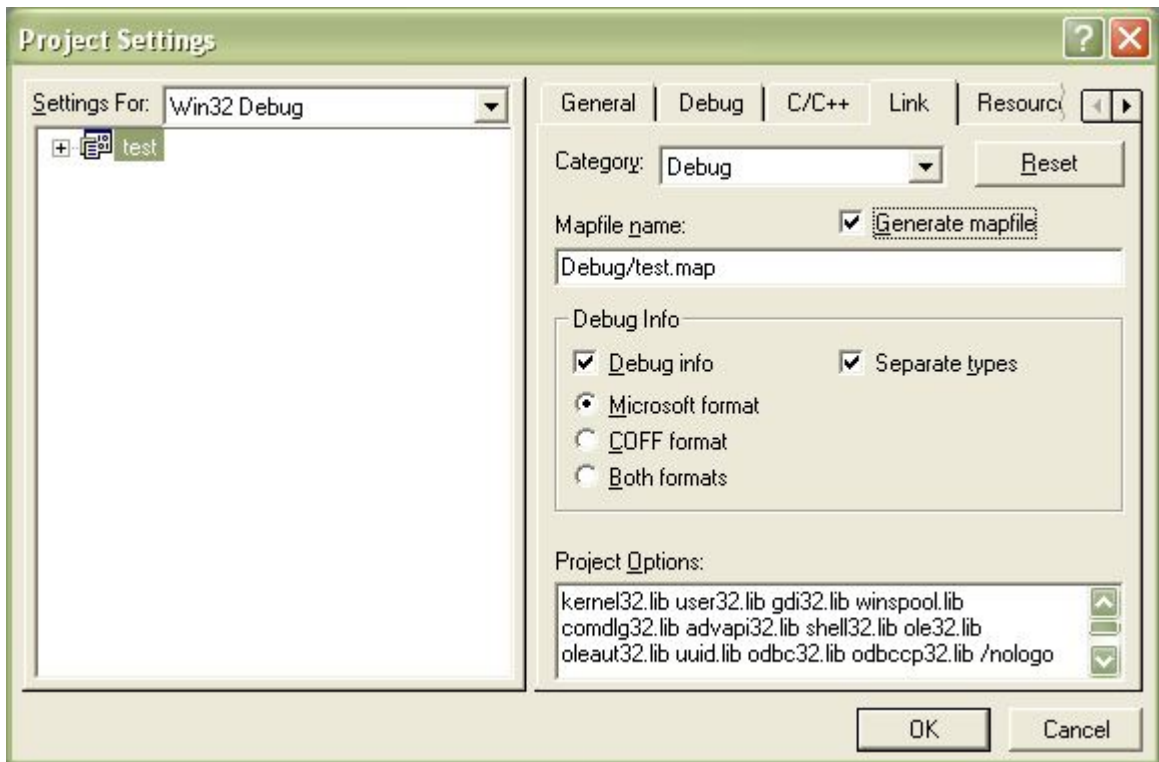
- Open the project of your module in MS Visual C++
- In the Main Menu select "Project" - "Settings" or press Alt + F7



- 1. Select "Link" tab and "Category" - "Debug"



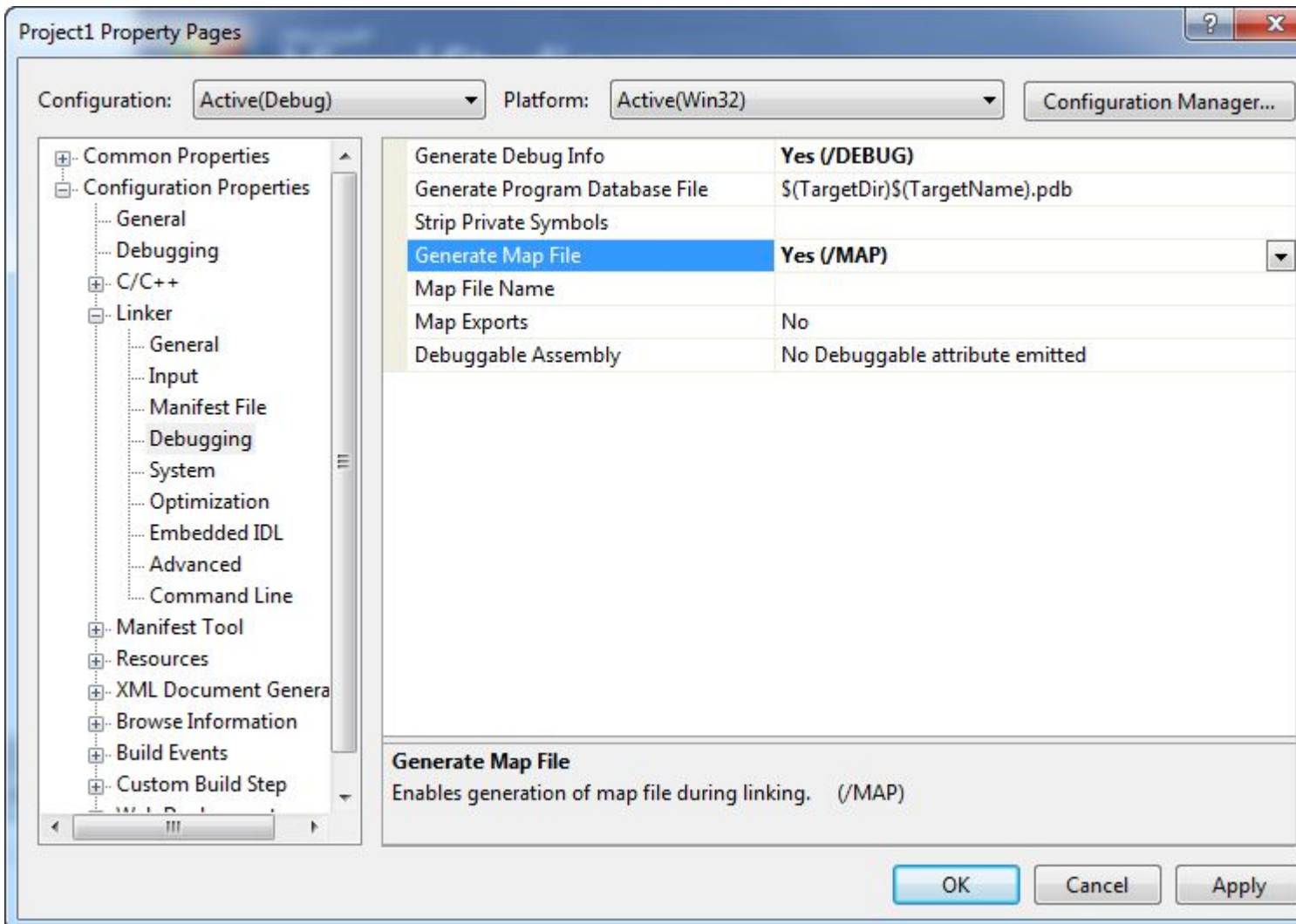
- 1. Enable the "Generate mapfile" option



- Compile your project.

Creating a map file in MS Visual Studio 2003:

- Open the project of your module in MS Visual C++;
- In the Main Menu select Project - Properties;
- In the appeared window select Configuration Properties - Linker - Debugging from the left navigation pane
- Generate Map File to Yes;



and map file.

Supported file formats

Enigma Protector supports any executable Windows 32/64 PE file, *.exe, *.dll, *.ocx, *.scr etc, files that are compiled with any kind of Win32/Win64 compiler, like C++, Delphi, MASM, Visual Basic, Free Pascal, FoxPro, Power Basic, TASM etc.

Work with the command line

Enigma Protector can be used to protect files through the command line. The command line of versions `enigma32.exe` or `enigma64.exe` is located in the installation folder.

- | To load the project file into Enigma Protector: `Enigma.exe Project.enigma`
- | To start the protection, use the console version:

Version	Example
Enigma Protector x86 (standard 32-bits version)	<code>enigma32.exe [options] Project.enigma</code>
Enigma Protector x64	<code>enigma64.exe [options] Project.enigma</code>

`Project.enigma` is the Enigma Protector project file.

The following options can be used:

- | `-q` enables quiet protection mode, all messages are suppressed
- | `-qe` enables quiet protection mode, all messages are suppressed except error messages

API Description

The Enigma API is a set of special functions that allow you to view the protection, trial and licensing parameters inside your application while the application is running. Please select the necessary API group:

- | [Registration API](#) - licensing and key validation functions;
- | [Trial API](#) - trial parameters and date/time monitoring functions;
- | [Crypt API](#) - hashing and encryption functions;
- | [Miscellaneous API](#) - protected strings, executed copies, watermarks, country, Enigma version, protection integrity and other miscellaneous functions;

See [How to use Enigma API](#) page to learn more about calling API from your application.

Warning: the Enigma API takes effect only after the module has been protected! Do not forget to protect the module before using it.

Registration API

Registration API is a set of functions that allow you to check, save, load and delete licensing information (registration name and key), extract additional information from the installed Registration key and get the current status of the Registration key and its expiration parameters.

- | EP_RegCheckKey
- | EP_RegCheckKeyA
- | EP_RegCheckKeyW
- | EP_RegLoadKey
- | EP_RegLoadKeyA
- | EP_RegLoadKeyW
- | EP_RegSaveKey
- | EP_RegSaveKeyA
- | EP_RegSaveKeyW
- | EP_RegDeleteKey
- | EP_RegLoadAndCheckKey
- | EP_RegCheckAndSaveKey
- | EP_RegCheckAndSaveKeyA
- | EP_RegCheckAndSaveKeyW
- | EP_RegHardwareID
- | EP_RegHardwareIDA
- | EP_RegHardwareIDW
- | EP_RegKeyExpirationDate
- | EP_RegKeyExpirationDateEx
- | EP_RegKeyCreationDate
- | EP_RegKeyCreationDateEx
- | EP_RegKeyExecutions
- | EP_RegKeyExecutionsLeft
- | EP_RegKeyExecutionsTotal
- | EP_RegKeyDays
- | EP_RegKeyDaysLeft
- | EP_RegKeyDaysTotal
- | EP_RegKeyRuntime
- | EP_RegKeyRuntimeLeft
- | EP_RegKeyRuntimeTotal
- | EP_RegKeyGlobalTime
- | EP_RegKeyGlobalTimeLeft
- | EP_RegKeyGlobalTimeTotal
- | EP_RegKeyRegisterAfterDate
- | EP_RegKeyRegisterAfterDateEx
- | EP_RegKeyRegisterBeforeDate
- | EP_RegKeyRegisterBeforeDateEx
- | EP_RegShowDialog

See [How to use Enigma API](#) page to learn more about calling API from your application.

Warning: the Enigma API takes effect only after the file has been protected! Do not forget to protect the file before using it.

EP_RegCheckKeyW

EP_RegCheckKeyW function serves for verifying registration information.

Parameters

- | `Name` - the registration name - a pointer to the null terminated ANSI string.
- | `Key` - the registration key - a pointer to the null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common panel](#).

The function fails in the following cases:

- | the registration information is incorrect;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegCheckKey( char* Name, char* Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

[Show/Hide C++ function example](#)

See function examples in the installation folder, Examples subfolder.

EP_RegCheckKeyA

EP_RegCheckKeyA function serves for verifying registration information. It is a duplicate and has the same functionality as the [EP_RegCheckKey](#) function.

Parameters

- | **Name** - the registration name - a pointer to the null terminated ANSI string.
- | **Key** - the registration key - a pointer to the null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common](#) panel.

The function fails in the following cases:

- | the registration information is incorrect;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegCheckKeyA( char* Name, char* Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples subfolder.

EP_RegCheckKey

EP_RegCheckKey function serves for verifying registration information. It has the same functionality as [EP_RegCheckKey](#), but is used for processing unicode (wide) strings data. Please note, to use this function you have to enable UNICODE Registration Scheme at [REGISTRATION FEATURES - Common panel](#).

Parameters

- | `Name` - the registration name - a pointer to the null terminated unicode (wide) string.
- | `Key` - the registration key - a pointer to the null terminated unicode (wide) string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration information is incorrect;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegCheckKeyW( wchar_t* Name, wchar_t* Key );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\RegistrationUnicode subfolder.

EP_RegCheckAndSaveKey

EP_RegCheckAndSaveKey function serves for verifying and saving the registration information. It combines couple functions [EP_RegCheckKey](#) and [EP_RegSaveKey](#). If the registration information is valid for the current project, then it will be saved, otherwise the function fails.

Parameters

- ▮ `Name` - the registration name - a pointer to the null terminated ANSI string.
- ▮ `Key` - the registration key - a pointer to the null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common panel](#).

The function fails in the following cases:

- ▮ the registration information is incorrect;
- ▮ an error occurred while saving the registration information (it can occur due to the enabled protection from writing files to the drive or entries in the registry);
- ▮ the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegCheckAndSaveKey( char* Name, char* Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

[Show/Hide C++ function example](#)

[Show/Hide C# \(.NET\) function example](#)

See function examples in the installation folder, Examples subfolder.

EP_RegCheckAndSaveKeyA

EP_RegCheckAndSaveKeyA function serves for verifying and saving the registration information. It is a duplicate and has the same functionality as the [EP_RegCheckAndSaveKey](#) function.

Parameters

- | **Name** - the registration name - a pointer to the null terminated ANSI string.
- | **Key** - the registration key - a pointer to the null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common panel](#).

The function fails in the following cases:

- | the registration information is incorrect;
- | an error occurred while saving the registration information (it can occur due to the enabled protection from writing files to the drive or entries in the registry);
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegCheckAndSaveKeyA( char* Name, char* Key );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples subfolder.

EP_RegCheckAndSaveKeyW

EP_RegCheckAndSaveKeyW function serves for verifying and saving the registration information. It has the same functionality as [EP_RegCheckAndSaveKey](#), but is used for processing unicode (wide) strings data. Please note, to use this function you should enable UNICODE Registration Scheme at [REGISTRATION FEATURES - Common panel](#).

Parameters

- | **Name** - the registration name - a pointer to the null terminated unicode (wide) string.
- | **Key** - the registration key - a pointer to the null terminated unicode (wide) string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration information is incorrect;
- | an error(s) occurred while saving the registration information (it can occur due to the enabled protection from writing files to the drive or entries in the registry);
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegCheckAndSaveKeyW( wchar_t* Name, wchar_t* Key );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\RegistrationUnicode subfolder.

EP_RegDeleteKey

EP_RegDeleteKey function serves for deleting the existing registration information.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | no registration information is present;
- | an error occurred while deleting the registration information (it can occur due to the enabled protection from deleting files from the drive or entries in the registry);
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegDeleteKey();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

[Show/Hide C++ function example](#)

See function examples in the installation folder, Examples subfolder.

EP_RegHardwareID

EP_RegHardware function serves for retrieving unique user PC information. See [Registration Features - Hardware Lock](#) panel for more detailed information.

Parameters

The function does not have parameters

Return Value

If the function succeeds, the return value is a pointer to the null terminated ANSI string. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common](#) panel.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall PCHAR EP_RegHardwareID();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

[Show/Hide C++ function example](#)

See function examples in the installation folder, Examples subfolder.

EP_RegHardwareIDA

EP_RegHardwareIDA function serves for retrieving unique user PC information. See [Registration Features - Hardware Lock](#) panel for more details.

Parameters

The function does not have parameters

Return Value

If the function succeeds, the return value is a pointer to the null terminated ANSI string. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common](#) panel.

Definition

Show/Hide C++ function definition

```
extern "C" __declspec( dllimport ) __stdcall PCHAR EP_RegHardwareIDA();
```

Show/Hide Delphi function definition

Show/Hide C# (.NET) function definition

See function examples in the installation folder, Examples subfolder.

EP_RegHardwareIDW

EP_RegHardwareIDW function serves for retrieving unique user PC information. See [Registration Features - Hardware Lock](#) panel for more details. Please note, to use this function you should enable UNICODE Registration Scheme at [REGISTRATION FEATURES - Common](#) panel.

Parameters

The function does not have parameters

Return Value

If the function succeeds, the return value is a pointer to the null terminated unicode (wide) string. If the function fails, the return value is 0.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall PWCHAR EP_RegHardwareIDW();
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\RegistrationUnicode subfolder.

EP_RegLoadKey

EP_RegLoadKey function serves for reading the registration information. The place and path where the registration information will be stored should be defined in [REGISTRATION FEATURES - Registration data storing](#) panel.

Parameters

- | **Name** - the registration name - a pointer that will return a null terminated ANSI string.
- | **Key** - the registration key - a pointer that will return a null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common](#) panel.

The function just reads the registration information, it does not verify key validation. To make sure you have the correct pair of registration name/key, you must manually check the key validation by means of [EP_RegCheckKey](#), or use the combine function [EP_RegLoadAndCheckKey](#).

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegLoadKey( char** Name, char** Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples subfolder.

EP_RegLoadKeyA

EP_RegLoadKey function serves for reading the registration information. It is a duplicate and has the same functionality as the [EP_RegLoadKey](#) function.

Parameters

- ▮ **Name** - the registration name - the pointer that will return a null terminated ANSI string.
- ▮ **Key** - the registration key - the pointer that will return a null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

UNICODE Registration Scheme should be disabled at [REGISTRATION FEATURES - Common panel](#).

The function just reads the registration information, it does not verify key validation. To make sure you have the correct pair of registration name/key, you must manually check the key validation by means of [EP_RegCheckKeyA](#), or use the combine function [EP_RegLoadAndCheckKey](#).

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegLoadKeyA( char** Name, char** Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples subfolder.

EP_RegLoadKeyW

EP_RegLoadKey function serves for reading the registration information. It has the same functionality as [EP_RegLoadKey](#) but is used for processing of unicode (wide) strings data. Please note, to use this function you have to enable UNICODE Registration Scheme at [REGISTRATION FEATURES - Common panel](#).

Parameters

- | `Name` - the registration name - the pointer that will return a null terminated unicode (wide) string.
- | `Key` - the registration key - the pointer that will return a null terminated unicode (wide) string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function just reads the registration information, it does not verify key validation. To make sure you have the correct pair of registration name/key, you must manually check the key validation by means of [EP_RegCheckKeyW](#), or use the combine function [EP_RegLoadAndCheckKey](#).

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegLoadKeyW( wchar_t** Name, wchar_t** Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\RegistrationUnicode subfolder.

EP_RegSaveKey

EP_RegSaveKey function serves for saving the registration information. The place and path where the registration information will be stored should be defined in [REGISTRATION FEATURES - Registration data storing](#) panel.

Parameters

- ▮ **Name** - the registration name - a pointer to the null terminated ANSI string.
- ▮ **Key** - the registration key - a pointer to the null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function only saves the registration information. To make sure you save the correct registration information, you should verify it by means of the [EP_RegCheckKey](#) function. Otherwise, use the combine function [EP_RegCheckAndSaveKey](#).

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegSaveKey( char* Name, char* Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

[Show/Hide C++ function example](#)

See function examples in the installation folder, Examples subfolder.

EP_RegSaveKeyA

EP_RegSaveKeyA function serves for saving the registration information. It is a duplicate and has the same functionality as the [EP_RegSaveKey](#) function.

Parameters

- 1 `Name` - the registration name - a pointer to the null terminated ANSI string.
- 1 `Key` - the registration key - a pointer to the null terminated ANSI string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function only saves the registration information. To make sure you save the correct registration information, you should verify it by means of the [EP_RegCheckKeyA](#) function. Otherwise, use the combine function [EP_RegCheckAndSaveKeyA](#).

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegSaveKeyA( char* Name, char* Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples subfolder.

EP_RegSaveKeyW

EP_RegSaveKeyW function serves for saving the registration information. It has the same functionality as [EP_RegSaveKey](#), but is used for processing the unicode (wide) strings data. Please note, to use this function you must enable UNICODE Registration Scheme at [REGISTRATION FEATURES - Common](#)

Parameters

- ▮ **Name** - the registration name - a pointer to the null terminated unicode (wide) string.
- ▮ **Key** - the registration key - a pointer to the null terminated unicode (wide) string.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function only saves the registration information. To make sure you save the correct registration information, you should verify it by means of the [EP_RegCheckKeyW](#) function. Otherwise, use the combine function [EP_RegCheckAndSaveKeyW](#).

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegSaveKeyW( wchar_t* Name, wchar_t* Key );
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\RegistrationUnicode subfolder.

EP_RegLoadAndCheckKey

EP_RegLoadAndCheckKey function serves for reading and verifying registration information. It combines couple functions [EP_RegLoadKey](#) and [EP_RegCheckKey](#).

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration information is not present;
- | the registration information is incorrect;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegLoadAndCheckKey();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

[\[+\] Show/Hide C++ function example](#)

See function examples in the installation folder, Examples subfolder.

EP_RegKeyCreationDate

EP_RegKeyCreationDate function serves for retrieving the Registration key creation date. See [Creating Keys](#) for more details. See also the extended function [EP_RegKeyCreationDateEx](#).

Parameters

- | **Year** - the year of the key creation date.
- | **Month** - the month of the key creation date.
- | **Day** - the day of the key creation date.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyCreationDate( int* Year, int* Month, int* Day );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

See function examples in the installation folder.

EP_RegKeyCreationDateEx

EP_RegKeyCreationDateEx returns the registration key creation date. See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is a 32-bit unsigned integer value that represents the Date value. The high byte of the high word contains a day value, the low byte of the high word contains a month value, the low word contains a year value. For example, the returned value 0x10012010 (\$10012010) means the 10 January 2010 date. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | there is no valid registration key;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyCreationDateEx();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyExpirationDate

EP_RegKeyExpirationDate function serves for retrieving the registration key expiration date. The registration should be generated with the Key Expiration Date option enabled. See [Creating Keys](#) for more details. See also the extended function [EP_RegKeyExpirationDateEx](#).

Parameters

- | `Year` - the year of the key expiration date.
- | `Month` - the month of the key expiration date.
- | `Day` - the day of the key expiration date.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | there is no valid registration key;
- | the registration key is not limited by time;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyExpirationDate( int* Year, int* Month, int* Day
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

See function examples in the installation folder, Examples\KeyExpiration subfolder.

EP_RegKeyExpirationDateEx

EP_RegKeyExpirationDateEx returns the expiration date of the registration key. The registration key should be generated with the Expiration Date option enabled. See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is a 32-bit unsigned integer value that represents the Date value. The high byte of the high word contains a day value, the low byte of the high word contains a month value, the low word contains a year value. For example, the returned value 0x10012010 (\$10012010) means the 10 January 2010 date. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | there is no valid registration key;
- | the registration key is not limited by time;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyExpirationDateEx();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\KeyExpiration subfolder.

EP_RegKeyExecutions

EP_RegKeyExecutions function returns the Total number of executions and the number of executions Left that the registration key is limited to. See [Creating Keys](#) for more details. See also the extended functions [EP_RegKeyExecutionsLeft](#) and [EP_RegKeyExecutionsTotal](#).

Parameters

- | `Total` - the total number of executions that the registration key is limited to.
- | `Left` - the number of executions left.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by executions;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyExecutions( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyExecutionsLeft

EP_RegKeyExecutionsLeft function returns the number of executions Left that the registration key is limited to. EP_RegKeyExecutionsLeft extends [EP_RegKeyExecutions](#). See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of executions Left. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by executions;
- | there is no valid registration key;
- | the executions limit has expired, there are no executions left;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyExecutionsLeft();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyExecutionsTotal

EP_RegKeyExecutionsTotal function returns the Total number of executions that the registration key is limited to. EP_RegKeyExecutionsTotal extends [EP_RegKeyExecutions](#). See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of Total executions. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by executions;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyExecutionsTotal();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyDays

EP_RegKeyDays function returns the Total number of days and the number of days Left that the registration key is limited to. See [Creating Keys](#) for more details. See also the extended functions [EP_RegKeyDaysLeft](#) and [EP_RegKeyDaysTotal](#).

Parameters

- | **Total** - the total number of days that the registration key is limited to.
- | **Left** - the number of days left.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by days;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyDays( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyDaysLeft

EP_RegKeyDaysLeft function returns the number of days Left that the registration key is limited to. EP_RegKeyDaysLeft extends [EP_RegKeyDays](#). See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of days Left. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by days;
- | there is no valid registration key;
- | the days limit has expired, there are no days left;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyDaysLeft();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyDaysTotal

EP_RegKeyDaysTotal function returns the Total number of days that the registration key is limited to. EP_RegKeyDaysTotal extends [EP_RegKeyDays](#). See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the Total number of days. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by days;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyDaysTotal();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyRuntime

EP_RegKeyRuntime function returns the Total number of Run-time minutes and the number of Run-time minutes Left that the registration key is limited to. See [Creating Keys](#) for more details. See also the extended functions [EP_RegKeyRuntimeLeft](#) and [EP_RegKeyRuntimeTotal](#).

Parameters

- | **Total** - the total number of run-time minutes that the registration key is limited to.
- | **Left** - the number of run-time minutes left.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by run-time;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyRuntime( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyRuntimeLeft

EP_RegKeyRuntimeLeft function returns the number of run-time minutes Left that the registration key is limited to. EP_RegKeyRuntimeLeft extends [EP_RegKeyRuntime](#). See [Creating Keys](#) for more detailed information.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of run-time minutes Left. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by the run-time;
- | there is no valid registration key;
- | the run-time limit has expired, there are no run-time minutes left;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyRuntimeLeft();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyRuntimeTotal

EP_RegKeyRuntimeTotal function returns the Total number of run-time minutes that the registration key is limited to. EP_RegKeyRuntimeTotal extends [EP_RegKeyRuntime](#). See [Creating Keys](#) for extended information.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of Total run-time minutes. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by the run-time;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyRuntimeTotal();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyGlobalTime

EP_RegKeyGlobalTime function returns the Total number of Global Time minutes and the number of Global Time minutes Left that the registration key is limited to. See [Creating Keys](#) for more details. See also the extended functions [EP_RegKeyGlobalTimeLeft](#) and [EP_RegKeyGlobalTimeTotal](#).

Parameters

- | `Total` - the total number of Global Time minutes that the registration key is limited to.
- | `Left` - the number of Global Time minutes left.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by Global Time;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyGlobalTime( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyGlobalTimeLeft

EP_RegKeyGlobalTimeLeft function returns the number of Global Time minutes Left that the registration key is limited to. EP_RegKeyGlobalTimeLeft extends [EP_RegKeyGlobalTime](#). See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of Global Time minutes Left. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by Global Time;
- | there is no valid registration key;
- | the Global Time limit has expired, there are no Global Time minutes left;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyGlobalTimeLeft();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyGlobalTimeTotal

EP_RegKeyGlobalTimeTotal function returns the Total number of Global Time minutes that the registration key is limited to. EP_RegKeyGlobalTimeTotal extends [EP_RegKeyGlobalTime](#). See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of Total Global Time minutes. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key is not limited by Global Time;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyGlobalTimeTotal();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyRegisterAfterDate

EP_RegKeyRegisterAfterDate returns the Register After date from the registration key. See [Creating Keys](#) for more details. See also the extended function [EP_RegKeyRegisterAfterDateEx](#).

Parameters

- | **Year** - the year of the key Register After date.
- | **Month** - the month of the key Register After date.
- | **Day** - the day of the key Register After date.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key does not contain the Register After date;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyRegisterAfterDate( int* Year, int* Month, int* D
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyRegisterAfterDateEx

EP_RegKeyRegisterAfterDateEx returns the Register After date from the registration key. See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is a 32-bit unsigned integer value that represents the Date value. The high byte of the high word contains a day value, the low byte of the high word contains a month value, the low word contains a year value. For example, the returned value 0x10012010 (\$10012010) means the 10 January 2010 date. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key does not contain the Register After date;
- | there is no valid registration key;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyRegisterAfterDateEx();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyRegisterBeforeDate

EP_RegKeyRegisterBeforeDate returns the Register Before date from the registration key. See [Creating Keys](#) for more. See also the extended function [EP_RegKeyRegisterBeforeDateEx](#).

Parameters

- | **Year** - the year of the key Register After date.
- | **Month** - the month of the key Register After date.
- | **Day** - the day of the key Register After date.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key does not contain the Register Before date;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_RegKeyRegisterBeforeDate( int* Year, int* Month, int* Day );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyRegisterBeforeDateEx

EP_RegKeyRegisterBeforeDateEx returns the Register Before date from the registration key. See [Creating Keys](#) for more details.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is a 32-bit unsigned integer value that represents the Date value. The high byte of the high word contains a day value, the low byte of the high word contains a month value, the low word contains a year value. For example, the returned value 0x10012010 (\$10012010) means the 10 January 2010 date. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the registration key does not contain the Register Before date;
- | there is no valid registration key;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyRegisterBeforeDateEx();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_RegKeyStatus

EP_RegKeyStatus returns the error status of registration information after the key verification routine. It should be called after any function that verifies registration information, for example, after [EP_RegCheckKey](#) or [EP_RegLoadAndCheckKey](#).

Parameters

The function does not have parameters.

Return Value

The function returns one of the following values:

Value	Description
KEY_STATUS_DOESNOTEXIST = 0;	Registration information was not found
KEY_STATUS_VALID = 1;	Registration information is valid
KEY_STATUS_INVALID = 2;	Registration information is invalid
KEY_STATUS_STOLEN = 3;	The registration key is marked as Stolen. See License Manager - Edit License
KEY_STATUS_DATEEXPIRED = 4;	The The registration key contains the Expiration date that has expired
KEY_STATUS_WITHOUTHARDWARELOCK = 5;	The registration key is not locked to hardware ID, but the application requires only hardware locked registration keys. See Key Properties
KEY_STATUS_WITHOUTEXPIRATIONDATE = 6;	The The registration key does not contain the expiration date, but the application requires only registration keys with the expiration date. See Key Properties
KEY_STATUS_WITHOUTREGISTERAFTERDATE = 7;	The The registration key does not contain the register after date, but the application requires only registration keys with the register after date. See Key Properties
KEY_STATUS_WITHOUTREGISTERBEFOREDATE = 8;	The The registration key does not contain the register before date, but the application requires only registration keys with the register before date. See Key Properties
KEY_STATUS_WITHOUTEXECUTIONSLIMIT = 9;	The The registration key does not contain the executions limit, but the application requires only registration keys with the executions limit. See Key Properties
KEY_STATUS_WITHOUTDAYSLIMIT = 10;	The The registration key does not contain the days limit, but the application requires only registration keys with the days limit. See Key Properties
KEY_STATUS_WITHOUTRUNTIMELIMIT = 11;	The The registration key does not contain the run-time limit, but the application requires only registration keys with the run-time limit. See Key Properties
KEY_STATUS_WITHOUTGLOBALTIMELIMIT = 12;	The The registration key does not contain the global time limit, but the application requires only registration keys with the global time limit. See Key Properties
KEY_STATUS_WITHOUTCOUNTRYLOCK = 13;	The registration key does not contain the country lock, but the application requires only registration keys with the country lock. See Key Properties
KEY_STATUS_COUNTRYINVALID = 14;	The The registration key is locked to the country that does not match the user's country
KEY_STATUS_REGISTERAFTERFAILED = 15;	The The registration key contains Register After date that has expired
KEY_STATUS_REGISTERBEFOREFAILED = 16;	The registration key contains Register Before date that has expired
KEY_STATUS_EXECUTIONSEXPIRED = 17;	The registration key contains Executions limit that has expired
KEY_STATUS_DAYSEXPIRED = 18;	The registration key contains Days limit that has expired
KEY_STATUS_RUNTIMEEXPIRED = 19;	The registration key contains Run-time limit that has expired
KEY_STATUS_GLOBALTIMEEXPIRED = 20;	The registration key contains Global Time limit that has expired
KEY_STATUS_HARDWARECHANGESEXCEEDED_VOLUMESERIAL = 21;	The registration key is locked to System Volume Serial hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock
KEY_STATUS_HARDWARECHANGESEXCEEDED_VOLUMENAME = 22;	The registration key is locked to System Volume Name hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock
KEY_STATUS_HARDWARECHANGESEXCEEDED_COMPUTERNAME = 23;	The registration key is locked to Computer Name hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock
KEY_STATUS_HARDWARECHANGESEXCEEDED_CPU = 24;	The registration key is locked to CPU Type hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock

KEY_STATUS_HARDWARECHANGESEXCEEDED_MOTHERBOARD = 25;	The registration key is locked to Motherboard hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock
KEY_STATUS_HARDWARECHANGESEXCEEDED_WINDOWSKEY = 26;	The registration key is locked to Windows Serial Key hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock
KEY_STATUS_HARDWARECHANGESEXCEEDED_HDDSERIAL = 27;	The registration key is locked to Hard Disk Serial Number hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock
KEY_STATUS_HARDWARECHANGESEXCEEDED_USERNAME = 28;	The registration key is locked to Windows User Name hardware ID that was changed, but the program does not allow changes of this hardware ID. See Hardware Lock

Remark

An unprotected application will always return KEY_STATUS_DOESNOTEXIST.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_RegKeyStatus();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples subfolder.

EP_RegKeyInformation

EP_RegKeyInformation extracts internal information from the registration key. Please note, this function cannot be used for registration, this function just extracts information from a particular pair of registration name and key. To verify the program, the functions like [EP_RegCheckKey](#), [EP_RegLoadAndCheckKey](#) and [EP_RegCheckAndSaveKey](#) should be used.

Parameters

- | **AName** - the registration name - a pointer to the null terminated ANSI string;
- | **AKey** - the registration key - a pointer to the null terminated ANSI string;
- | **AKeyInfo** - TKeyInformation - a pointer to the TKeyInformation structure.

TKeyInformation contains the following members:

- | **Stolen** - indicates if the registration key is stolen. See [License Manager - Edit License](#)
- | **CreationYear** - the key creation year
- | **CreationMonth** - the key creation month
- | **CreationDay** - the key creation day
- | **UseKeyExpiration** - indicates if the registration key contains the expiration date
- | **ExpirationYear** - the key expiration year
- | **ExpirationMonth** - the key expiration month
- | **ExpirationDay** - the key expiration day
- | **UseHardwareLocking** - indicates if the registration key is hardware locked
- | **UseExecutionsLimit** - shows if the registration key contains the executions limit
- | **ExecutionsCount** - the number of executions
- | **UseDaysLimit** - shows if the registration key contains the days limit
- | **DaysCount** - the number of days
- | **UseRunTimeLimit** - shows if the registration key contains the run-time limit
- | **RunTimeMinutes** - the number of run-time minutes
- | **UseGlobalTimeLimit** - shows if the registration key contains the Global Time limit
- | **GlobalTimeMinutes** - the number of Global Time minutes
- | **UseCountryLimit** - shows if the registration key contains the country lock
- | **CountryCode** - the country code
- | **UseRegisterAfter** - shows if the registration key contains the register after date
- | **RegisterAfterYear** - the year of the register after date
- | **RegisterAfterMonth** - the month of the register after date
- | **RegisterAfterDay** - the day of the register after date
- | **UseRegisterBefore** - shows if the registration key contains the register before date
- | **RegisterBeforeYear** - the year of the register before date
- | **RegisterBeforeMonth** - the month of the register before date
- | **RegisterBeforeDay** - the month of the register before date
- | **EncryptedSections** - a 16-value array of encrypted sections that the registration key decrypts

Return Value

If the function fails, the return value is 0. If the function succeeds, the return value is not zero and the TKeyInformation contains the information extracted from the passed registration key.

Remark

The function fails in the following cases:

- | the registration information is incorrect;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
typedef struct TKeyInformation {
    BOOL Stolen;           // {out} is key stolen
    DWORD CreationYear;   // {out} key creation year
    DWORD CreationMonth;  // {out} key creation month
    DWORD CreationDay;    // {out} key creation day
```

```

BOOL UseRunTimeLimit; // {out} limit key by run time?
DWORD RunTimeMinutes; // {out} run time minutes
BOOL UseGlobalTimeLimit; // {out} limit key by global time?
DWORD GlobalTimeMinutes; // {out} global time minutes
BOOL UseCountyLimit; // {out} limit key by country?
DWORD CountryCode; // {out} country code
BOOL UseRegisterAfter; // {out} register key after date?
DWORD RegisterAfterYear; // {out} register after year
DWORD RegisterAfterMonth; // {out} register after month
DWORD RegisterAfterDay; // {out} register after day
BOOL UseRegisterBefore; // {out} register key before date?
DWORD RegisterBeforeYear; // {out} register before year
DWORD RegisterBeforeMonth; // {out} register before month
DWORD RegisterBeforeDay; // {out} register before day
BOOL EncryptedSections[NUMBER_OF_CRYPTED_SECTIONS]; // {out} Crypted sections
} TKeyInformation, *PKeyInformation;

extern "C" __declspec( dllexport ) __stdcall BOOL EP_RegKeyInformation(char* AName, char* AKey, PKeyInformation

```

- [⊕ Show/Hide Delphi function definition](#)
- [⊕ Show/Hide Visual Basic function definition](#)
- [⊕ Show/Hide C# \(.NET\) function definition](#)

Examples

- [⊕ Show/Hide Delphi function example](#)
- [⊕ Show/Hide C++ function example](#)
- [⊕ Show/Hide C# \(.NET\) function example](#)

See function examples in the installation folder, Examples\KeyInformation subfolder.

EP_RegKeyInformationA

EP_RegKeyInformationA extracts internal information from the registration key. Please note, this function cannot registration, this function just extracts information from a particular pair of registration name and key. To verify the pro the functions like [EP_RegCheckKey](#), [EP_RegLoadAndCheckKey](#) and [EP_RegCheckAndSaveKey](#) should be used.

Parameters

- | **AName** - the registration name - a pointer to the null terminated ANSI string;
- | **AKey** - the registration key - a pointer to the null terminated ANSI string;
- | **AKeyInfo** - TKeyInformation - a pointer to the TKeyInformation structure.

TKeyInformation contains the following members:

- | **Stolen** - indicates if the registration key is stolen. See [License Manager - Edit License](#)
- | **CreationYear** - the key creation year
- | **CreationMonth** - the key creation month
- | **CreationDay** - the key creation day
- | **UseKeyExpiration** - indicates if the registration key contains the expiration date
- | **ExpirationYear** - the key expiration year
- | **ExpirationMonth** - the key expiration month
- | **ExpirationDay** - the key expiration day
- | **UseHardwareLocking** - indicates if the registration key is hardware locked
- | **UseExecutionsLimit** - shows if the registration key contains the executions limit
- | **ExecutionsCount** - the number of executions
- | **UseDaysLimit** - shows if the registration key contains the days limit
- | **DaysCount** - the number of days
- | **UseRunTimeLimit** - shows if the registration key contains the run-time limit
- | **RunTimeMinutes** - the number of run-time minutes
- | **UseGlobalTimeLimit** - shows if the registration key contains the Global Time limit
- | **GlobalTimeMinutes** - the number of Global Time minutes
- | **UseCountyLimit** - shows if the registration key contains the country lock
- | **CountryCode** - the country code
- | **UseRegisterAfter** - shows if the registration key contains the register after date
- | **RegisterAfterYear** - the year of the register after date
- | **RegisterAfterMonth** - the month of the register after date
- | **RegisterAfterDay** - the day of the register after date
- | **UseRegisterBefore** - shows if the registration key contains the register before date
- | **RegisterBeforeYear** - the year of the register before date
- | **RegisterBeforeMonth** - the month of the register before date
- | **RegisterBeforeDay** - the month of the register before date
- | **EncryptedSections** - a 16-value array of encrypted sections that the registration key decrypts

Return Value

If the function fails, the return value is 0. If the function succeeds, the return value is not zero and the TKeyInformation information extracted from the passed registration key.

Remark

The function fails in the following cases:

- | the registration information is incorrect;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
typedef struct TKeyInformation {
    BOOL Stolen;           // {out} is key stolen
    DWORD CreationYear;   // {out} key creation year
    DWORD CreationMonth;  // {out} key creation month
    DWORD CreationDay;    // {out} key creation day
```

```

    BOOL UseRunTimeLimit; // {out} limit key by run time?
    DWORD RunTimeMinutes; // {out} run time minutes
    BOOL UseGlobalTimeLimit; // {out} limit key by global time?
    DWORD GlobalTimeMinutes; // {out} global time minutes
    BOOL UseCountyLimit; // {out} limit key by country?
    DWORD CountryCode; // {out} country code
    BOOL UseRegisterAfter; // {out} register key after date?
    DWORD RegisterAfterYear; // {out} register after year
    DWORD RegisterAfterMonth; // {out} register after month
    DWORD RegisterAfterDay; // {out} register after day
    BOOL UseRegisterBefore; // {out} register key before date?
    DWORD RegisterBeforeYear; // {out} register before year
    DWORD RegisterBeforeMonth; // {out} register before month
    DWORD RegisterBeforeDay; // {out} register before day
    BOOL EncryptedSections[NUMBER_OF_CRYPTED_SECTIONS]; // {out} Crypted sections
} TKeyInformation, *PKeyInformation;

extern "C" __declspec( dllexport ) __stdcall BOOL EP_RegKeyInformationA(char* AName, char* AKey, PKeyInforma

```

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\KeyInformation subfolder.

EP_RegKeyInformationW

EP_RegKeyInformationW extracts internal information from the registration key. Please note, this function cannot be used to extract information from a particular pair of registration name and key. To verify the program registration information, [EP_RegCheckKey](#), [EP_RegLoadAndCheckKey](#) and [EP_RegCheckAndSaveKey](#) should be used.

Parameters

- | **AName** - the registration name - a pointer to the null terminated wide string;
- | **AKey** - the registration key - a pointer to the null terminated wide string;
- | **AKeyInfo** - TKeyInformation - a pointer to the TKeyInformation structure.

TKeyInformation contains the following members:

- | **Stolen** - indicates if the registration key is stolen. See [License Manager - Edit License](#)
- | **CreationYear** - the key creation year
- | **CreationMonth** - the key creation month
- | **CreationDay** - the key creation day
- | **UseKeyExpiration** - indicates if the registration key contains the expiration date
- | **ExpirationYear** - the key expiration year
- | **ExpirationMonth** - the key expiration month
- | **ExpirationDay** - the key expiration day
- | **UseHardwareLocking** - indicates if the registration key is hardware locked
- | **UseExecutionsLimit** - shows if the registration key contains the executions limit
- | **ExecutionsCount** - the number of executions
- | **UseDaysLimit** - shows if the registration key contains the days limit
- | **DaysCount** - the number of days
- | **UseRunTimeLimit** - shows if the registration key contains the run-time limit
- | **RunTimeMinutes** - the number of run-time minutes
- | **UseGlobalTimeLimit** - shows if the registration key contains the Global Time limit
- | **GlobalTimeMinutes** - the number of the Global Time minutes
- | **UseCountryLimit** - shows if the registration key contains the country lock
- | **CountryCode** - the country code
- | **UseRegisterAfter** - shows if the registration key contains the register after date
- | **RegisterAfterYear** - the year of the register after date
- | **RegisterAfterMonth** - the month of the register after date
- | **RegisterAfterDay** - the day of the register after date
- | **UseRegisterBefore** - shows if the registration key contains the register before date
- | **RegisterBeforeYear** - the year of the register before date
- | **RegisterBeforeMonth** - the month of the register before date
- | **RegisterBeforeDay** - the day of the register before date
- | **EncryptedSections** - a 16-value array of encrypted sections that the registration key decrypts

Return Value

If the function fails, the return value is 0. If the function succeeds, the return value is not zero and the TKeyInformation structure contains the information extracted from the passed registration key.

Remark

The function fails in the following cases:

- | the registration information is incorrect;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
typedef struct TKeyInformation {
    BOOL Stolen;           // {out} is key stolen
    DWORD CreationYear;   // {out} key creation year
    DWORD CreationMonth;  // {out} key creation month
    DWORD CreationDay;    // {out} key creation day
```

```
    BOOL UseCountyLimit;           // {out} limit key by country?
    DWORD CountryCode;            // {out} country code
    BOOL UseRegisterAfter;        // {out} register key after date?
    DWORD RegisterAfterYear;      // {out} register after year
    DWORD RegisterAfterMonth;     // {out} register after month
    DWORD RegisterAfterDay;       // {out} register after day
    BOOL UseRegisterBefore;       // {out} register key before date?
    DWORD RegisterBeforeYear;     // {out} register before year
    DWORD RegisterBeforeMonth;    // {out} register before month
    DWORD RegisterBeforeDay;      // {out} register before day
    BOOL EncryptedSections[NUMBER_OF_CRYPTED_SECTIONS]; // {out} Crypted sections
} TKeyInformation, *PKeyInformation;

extern "C" __declspec( dllexport ) __stdcall BOOL EP_RegKeyInformationW(wchar_t* AName, wchar_t* AKey, PKeyInformation);
```

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\KeyInformation subfolder.

EP_RegShowDialog

EP_RegShowDialog shows a registration dialog that is designed and enabled on the [REGISTRATION FEATURES - Registration Dialog](#) panel. Note that the Registration Dialog feature should be enabled, otherwise the dialog will not be shown. Also, any Exit application actions in the Registration Dialog will not exit the application, but just close the dialog window.

Return Value

The function does not return any value.

Remark

The function fails in the following cases:

- | the application is not protected;
- | [REGISTRATION FEATURES - Registration Dialog](#) feature is not enabled.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall void EP_RegShowDialog();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

See function examples in the installation folder, Examples\RegistrationShowDialog subfolder.

Trial API

Trial API is a set of functions that allow you to check trial period parameters and expiration.

- | [EP_TrialExecutions](#)
- | [EP_TrialExecutionsLeft](#)
- | [EP_TrialExecutionsTotal](#)
- | [EP_TrialDays](#)
- | [EP_TrialDaysLeft](#)
- | [EP_TrialDaysTotal](#)
- | [EP_TrialExpirationDate](#)
- | [EP_TrialExpirationDateEx](#)
- | [EP_TrialDateTillDate](#)
- | [EP_TrialDateTillDateStartEx](#)
- | [EP_TrialDateTillDateEndEx](#)
- | [EP_TrialExecutionTime](#)
- | [EP_TrialExecutionTimeLeft](#)
- | [EP_TrialExecutionTimeTotal](#)

See [How to use Enigma API](#) page to learn more about calling API from your application.

Warning: the Enigma API takes effect only after the file has been protected! Do not forget to protect the file before using it.

EP_TrialExecutions

EP_TrialExecutions function servers for retrieving the total trial period executions and the trial period executions left. The total executions count of the trial period should be defined in [TRIAL CONTROL - Limitation of executions count](#) panel. See also the extended functions [EP_TrialExecutionsLeft](#) and [EP_TrialExecutionsTotal](#).

Parameters

- | **Total** - the total number of trial executions.
- | **Left** - the number of trial executions left.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the executions count was not enabled;
- | the application is not protected.

If the user's PC has several user accounts, the trial information will be different for each user.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_TrialExecutions( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

[\[+\] Show/Hide C++ function example](#)

See function examples in the installation folder, Examples\Trial subfolder.

EP_TrialExecutionsLeft

EP_TrialExecutionsLeft function returns the number of trial executions Left. EP_TrialExecutionsLeft extends [EP_TrialExecutions](#).

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of executions Left. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the executions count was not enabled;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialExecutionsLeft();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialExecutionsTotal

EP_TrialExecutionsTotal function returns the Total number of trial executions. EP_TrialExecutionsTotal extends [EP_TrialExecutions](#).

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of Total executions. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- the limitation of the executions count was not enabled;
- the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialExecutionsTotal();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialDays

EP_TrialDays function servers for retrieving the total number of trial period days and thenumber of trial period days left. The total days count of the trial period should be defined in [TRIAL CONTROL - Limitation of days count](#) panel. See also the extended functions [EP_TrialDaysLeft](#) and [EP_TrialDaysTotal](#).

Parameters

- | **Total** - the total number of trial days.
- | **Left** - the number of trial days left.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the days count was not enabled;
- | the application is not protected.

If the user's PC has several user accounts, the trial information will be different for each user.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_TrialDays( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

[\[+\] Show/Hide C++ function example](#)

See function examples in the installation folder, Examples\Trial subfolder.

EP_TrialDaysLeft

EP_TrialDaysLeft function returns the number of trial days Left. EP_TrialDaysLeft extends [EP_TrialDays](#) function.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of days Left. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the days count was not enabled;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialDaysLeft();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialDaysTotal

EP_TrialDaysTotal function returns the Total number of trial days. EP_TrialDaysTotal extends [EP_TrialDays](#) function.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of Total days. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the days count was not enabled;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialDaysTotal();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialExpirationDate

EP_TrialExpirationDate function returns the trial expiration date. The trial expiration date should be defined on the [TRIAL CONTROL - Limitation of expiration date](#) panel. See also the extended function [EP_TrialExpirationDateEx](#).

Parameters

- | **Year** - the year of the trial expiration date.
- | **Month** - the month of the trial expiration date.
- | **Day** - the day of the trial expiration date.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the expiration date was not enabled;
- | the application is not protected.

If the user's PC has several user accounts, the trial information will be different for each user.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_TrialExpirationDate( int* Year, int* Month, int* Day )
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

See function examples in the installation folder, Examples\Trial subfolder.

EP_TrialExpirationDateEx

EP_TrialExpirationDateEx function returns the trial expiration date. It extends [EP_TrialExpirationDate](#) function.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is a 32-bit unsigned integer value that represents the Date value. The high byte of the high word contains a day value, the low byte of the high word contains a month value, the low word contains a year value. For example, the returned value 0x10012010 (\$10012010) means the 10 January 2010 date. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the expiration date was not enabled;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialExpirationDateEx();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialDateTillDate

EP_TrialDateTillDate function serves for retrieving start and end trial dates. Trial dates should be defined on the [TRIAL](#) extended functions [EP_TrialDateTillDateStartEx](#) and [EP_TrialDateTillDateEndEx](#).

Parameters

- | `StartYear` - the year of the start trial date.
- | `StartMonth` - the month of the start trial date.
- | `StartDay` - the day of the start trial date.
- | `EndYear` - the year of the end trial date.
- | `EndMonth` - the month of the end trial date.
- | `EndDay` - the day of the end trial date.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the expiration date was not enabled;
- | the application is not protected.

If the user's PC has several user accounts, the trial information will differ for each user.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_TrialDateTillDate( int* StartYear, int* StartMonth, int
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Trial subfolder.

EP_TrialDateTillDateStartEx

EP_TrialDateTillDateStartEx function returns the Start Date of the trial [Limitation Date till Date](#). It extends [EP_TrialDateTillDate](#) function.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is a 32-bit unsigned integer value that represents the Date value. The high byte of the high word contains a day value, the low byte of the high word contains a month value, the low word contains a year value. For example, the returned value 0x10012010 (\$10012010) means the 10 January 2010 date. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- the limitation of the expiration date was not enabled;
- the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialDateTillDateStartEx();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialDateTillDateEndEx

EP_TrialDateTillDateEndEx function returns the End Date of trial [Limitation Date till Date](#). It extends [EP_TrialDateTillDate](#) function.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is a 32-bit unsigned integer value that represents the Date value. The high byte of the high word contains a day value, the low byte of the high word contains a month value, the low word contains a year value. For example, the returned value 0x10012010 (\$10012010) means the 10 January 2010 date. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the expiration date was not enabled;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialDateTillDateEndEx();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialExecutionTime

EP_TrialExecutionTime function servers for retrieving the total trial period minutes and the trial period minutes left since the module start. The total number of trial minutes should be defined in [TRIAL CONTROL - Limitation of execution time](#) panel. See also the extended functions [EP_TrialExecutionTimeLeft](#) and [EP_TrialExecutionTimeTotal](#).

Parameters

- | `Total` - the total number of trial minutes.
- | `Left` - the number of trial minutes left.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | the limitation of the execution time was not enabled;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_TrialExecutionTime( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

[\[+\] Show/Hide C++ function example](#)

See function examples in the installation folder, Examples\TrialExecutionTime subfolder.

EP_TrialExecutionTimeLeft

EP_TrialExecutionTimeLeft function returns the number of trial execution minutes Left. EP_TrialExecutionTimeLeft extends [EP_TrialExecutionTime](#).

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of trial execution minutes Left. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- the limitation of the execution time was not enabled;
- the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialExecutionTimeLeft();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_TrialExecutionTimeTotal

EP_TrialExecutionTimeTotal function returns the Total number of trial execution minutes. EP_TrialExecutionTimeTotal extends [EP_TrialExecutionTime](#).

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of Total trial execution minutes. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- the limitation of the execution time was not enabled;
- the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_TrialExecutionTimeTotal();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

Crypt API

Crypt API is a set of functions that allow you to generate a hash (checksum) of the strings, files or custom buffer and perform buffer encryption/decryption.

- | [EP_CryptHashBuffer](#)
- | [EP_CryptHashFileA](#)
- | [EP_CryptHashFileW](#)
- | [EP_CryptHashStringA](#)
- | [EP_CryptHashStringW](#)
- | [EP_CryptDecryptBuffer](#)
- | [EP_CryptDecryptBufferEx](#)
- | [EP_CryptEncryptBuffer](#)
- | [EP_CryptEncryptBufferEx](#)

See [How to use Enigma API](#) page to learn more about calling API from your application.

Warning: the Enigma API takes effect only after the file has been protected! Do not forget to protect the file before using it.

EP_CryptDecryptBuffer

EP_CryptDecryptBuffer decrypts a buffer with the defined key.

Parameters

- | `Buffer` - a pointer to the memory buffer for decrypting.
- | `Size` - the size of the memory buffer.
- | `Key` - a pointer to ANSI, a null terminated string that will be used as a key for decryption.

Return Value

The function does not return value.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall void EP_CryptDecryptBuffer( byte* Buffer, int Size, char* Key)
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\CryptBuffer subfolder.

EP_CryptDecryptBufferEx

EP_CryptDecryptBufferEx decrypts a buffer with the defined key.

Parameters

- | `InBuffer` - a pointer to the input memory buffer for decrypting.
- | `OutBuffer` - a pointer to the output memory buffer for decrypting.
- | `Size` - the size of the memory buffer.
- | `Key` - a pointer to the buffer that will be used as a key for decryption.
- | `KeySize` - the size of the key memory buffer.

Return Value

The function does not return value.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall void EP_CryptDecryptBufferEx( byte* InBuffer, byte* OutBuffer,
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\CryptBuffer subfolder.

EP_CryptEncryptBuffer

EP_CryptEncryptBuffer encrypts a buffer with the defined key.

Parameters

- | `Buffer` - a pointer to the memory buffer for encrypting.
- | `Size` - the size of the memory buffer.
- | `Key` - a pointer to ANSI, a null terminated string that will be used as a key for encryption.

Return Value

The function does not return value.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall void EP_CryptEncryptBuffer( byte* Buffer, int Size, char* Key)
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\CryptBuffer subfolder.

EP_CryptEncryptBufferEx

EP_CryptEncryptBufferEx encrypts a buffer with the defined key.

Parameters

- | `InBuffer` - a pointer to the input memory buffer for encrypting.
- | `OutBuffer` - a pointer to the output memory buffer for encrypting.
- | `Size` - the size of the memory buffer.
- | `Key` - a pointer to the buffer that will be used as a key for encryption.
- | `KeySize` - the size of the key memory buffer.

Return Value

The function does not return value.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall void EP_CryptEncryptBufferEx( byte* InBuffer, byte* OutBuffer,
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\CryptBuffer subfolder.

EP_CryptHashBuffer

EP_CryptHashBuffer calculates hash of the user defined buffer.

Parameters

- | **Hash** - the type of the hash for calculation.

Hash Type	Parameter = Value
XOR32	HASH_XOR32 = 0
MD2	HASH_MD2 = 1
MD5	HASH_MD5 = 2
RipeMD160	HASH_RipeMD160 = 3
SH1	HASH_SH1 = 4
SHA224	HASH_SHA224 = 5
SHA256	HASH_SHA256 = 6
SHA384	HASH_SHA384 = 7
SHA512	HASH_SHA512 = 8

- | **Buffer** - a pointer to the memory buffer for hash calculating.
- | **Size** - the size of the memory buffer.
- | **Digest** - a pointer to the buffer for storing the hash value.

Return Value

If the function succeeds, the return value is the size of hash (that is stored in Digest buffer) in bytes. If the function fails, the return value is 0.

Hash Type	Hash Size (bytes)
XOR32	4
MD2	16
MD5	16
RipeMD160	20
SH1	20
SHA224	28
SHA256	32
SHA384	48
SHA512	64

Remark

The function fails in the following cases:

- | the input buffer is not allocated;
- | the input buffer is not readable;
- | Digest buffer is not allocated;
- | Digest buffer is write protected;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_CryptHashBuffer( int Hash, byte* Buffer, int Size, byte*
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Hashes subfolder.

EP_CryptHashFileA

EP_CryptHashFileA calculates hash of the user defined file.

Parameters

- | **Hash** - the type of the hash for calculation.

Hash Type	Parameter = Value
XOR32	HASH_XOR32 = 0
MD2	HASH_MD2 = 1
MD5	HASH_MD5 = 2
RipeMD160	HASH_RipeMD160 = 3
SH1	HASH_SH1 = 4
SHA224	HASH_SHA224 = 5
SHA256	HASH_SHA256 = 6
SHA384	HASH_SHA384 = 7
SHA512	HASH_SHA512 = 8

- | **FileName**- the name of the file, a pointer to the null terminated ANSI string.
- | **Digest** - a pointer to the buffer for storing the hash value.

Return Value

If the function succeeds, the return value is the size of hash (that is stored in Digest buffer) in bytes. If the function fails the return value is 0.

Hash Type	Hash Size (bytes)
XOR32	4
MD2	16
MD5	16
RipeMD160	20
SH1	20
SHA224	28
SHA256	32
SHA384	48
SHA512	64

Remark

The function fails in the following cases:

- | FileName is not set;
- | FileName is not readable;
- | Digest buffer is not allocated;
- | Digest buffer is write protected;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_CryptHashFileA( int Hash, char* FileName, byte* Digest)
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Hashes subfolder.

EP_CryptHashFileW

EP_CryptHashFileA calculates hash of the user defined file.

Parameters

- Hash - the type of the hash for calculation.

Hash Type	Parameter = Value
XOR32	HASH_XOR32 = 0
MD2	HASH_MD2 = 1
MD5	HASH_MD5 = 2
RipeMD160	HASH_RipeMD160 = 3
SH1	HASH_SH1 = 4
SHA224	HASH_SHA224 = 5
SHA256	HASH_SHA256 = 6
SHA384	HASH_SHA384 = 7
SHA512	HASH_SHA512 = 8

- FileName - the name of the file, a pointer to the null terminated UNICODE string.
- Digest - a pointer to the buffer for storing the hash value.

Return Value

If the function succeeds, the return value is the size of hash (that is stored in Digest buffer) in bytes. If the function fails, return value is 0.

Hash Type	Hash Size (bytes)
XOR32	4
MD2	16
MD5	16
RipeMD160	20
SH1	20
SHA224	28
SHA256	32
SHA384	48
SHA512	64

Remark

The function fails in the following cases:

- FileName is not set;
- FileName is not readable;
- Digest buffer is not allocated;
- Digest buffer is write protected;
- the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_CryptHashFileW( int Hash, wchar_t* FileName, byte* Digest );
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Hashes subfolder.

EP_CryptHashStringA

EP_CryptHashStringA calculates hash of the user defined ANSI string.

Parameters

- Hash - the type of the hash for calculation.

Hash Type	Parameter = Value
XOR32	HASH_XOR32 = 0
MD2	HASH_MD2 = 1
MD5	HASH_MD5 = 2
RipeMD160	HASH_RipeMD160 = 3
SH1	HASH_SH1 = 4
SHA224	HASH_SHA224 = 5
SHA256	HASH_SHA256 = 6
SHA384	HASH_SHA384 = 7
SHA512	HASH_SHA512 = 8

- Str - a pointer to the null terminated ANSI string.
- Digest - a pointer to the buffer for storing the hash value.

Return Value

If the function succeeds, the return value is the size of hash (that is stored in Digest buffer) in bytes. If the function fails, the return value is 0.

Hash Type	Hash Size (bytes)
XOR32	4
MD2	16
MD5	16
RipeMD160	20
SH1	20
SHA224	28
SHA256	32
SHA384	48
SHA512	64

Remark

The function fails in the following cases:

- Str is not set;
- Digest buffer is not allocated;
- Digest buffer is write protected;
- the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_CryptHashStringA( int Hash, char* Str, byte* Digest);
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Hashes subfolder.

EP_CryptHashStringW

EP_CryptHashFileW calculates hash of a user defined UNICODE string.

Parameters

- Hash - the type of the hash for calculation.

Hash Type	Parameter = Value
XOR32	HASH_XOR32 = 0
MD2	HASH_MD2 = 1
MD5	HASH_MD5 = 2
RipeMD160	HASH_RipeMD160 = 3
SH1	HASH_SH1 = 4
SHA224	HASH_SHA224 = 5
SHA256	HASH_SHA256 = 6
SHA384	HASH_SHA384 = 7
SHA512	HASH_SHA512 = 8

- Str - a pointer to the null terminated UNICODE string.
- Digest - a pointer to the buffer for storing the hash value.

Return Value

If the function succeeds, the return value is the size of hash (that is stored in Digest buffer) in bytes. If the function fails the return value is 0.

Hash Type	Hash Size (bytes)
XOR32	4
MD2	16
MD5	16
RipeMD160	20
SH1	20
SHA224	28
SHA256	32
SHA384	48
SHA512	64

Remark

The function fails in the following cases:

- Str is not set;
- Digest buffer is not allocated;
- Digest buffer is write protected;
- the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_CryptHashStringW( int Hash, wchar_t* Str, byte* Digest )
```

[Show/Hide Delphi function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Hashes subfolder.

Miscellaneous API

Miscellaneous API is a set of functions that provide additional functionality such as getting protected strings, checking the number of executed copies, extracting watermarks, detecting the country code, getting the Enigma version, checking protection integrity and other miscellaneous functions

- | [EP_MiscCountryCode](#)
- | [EP_MiscGetWatermark](#)
- | [EP_ProtectedStringByID](#)
- | [EP_ProtectedStringByKey](#)
- | [EP_CheckupCopies](#)
- | [EP_CheckupCopiesCurrent](#)
- | [EP_CheckupCopiesTotal](#)
- | [EP_CheckupIsEnigmaOk](#)
- | [EP_CheckupIsProtected](#)
- | [EP_CheckupVirtualizationTools](#)

See [How to use Enigma API](#) page to learn more about calling API from your application.

Warning: the Enigma API takes effect only after the file has been protected! Do not forget to protect the file before using it.

EP_CheckupCopies

EP_CheckupCopies function returns the total and current number of executed copies of the protected file. The initial settings should be entered in the [CHECK-UP - Executed Copies](#) panel. Also, see the extended functions [EP_CheckupCopiesCurrent](#) and [EP_CheckupCopiesTotal](#).

Parameters

- | **Total** - the total number of executed copies allowed.
- | **Current** - the current number of executed copies.

Return Value

If the function succeeds, the return value is 1. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | checkup of executed copies is not enabled;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_TrialDays( int* Total, int* Left );
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

See function examples in the installation folder, Examples\ExecutedCopies subfolder.

EP_CheckupCopiesCurrent

EP_CheckupCopiesCurrent function returns the current number of executed copies of the protected application. EP_CheckupCopiesCurrent extends the [EP_CheckupCopies](#) function.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the number of executed copies of the current application. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | backup of executed copies is not enabled;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_CheckupCopiesCurrent();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_CheckupCopiesTotal

EP_CheckupCopiesTotal function returns the total number of allowed simultaneously executed copies of the protected application. EP_CheckupCopiesTotal extends the [EP_CheckupCopies](#) function.

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the total number of allowed executed copies. If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | checkup of executed copies is not enabled;
- | the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_CheckupCopiesTotal();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder.

EP_CheckupIsEnigmaOk

EP_CheckupIsEnigmaOk function checks the integrity of the protection code. It is a useful function that can check the control sum of Enigma loader and protection itself, and if there are any errors, i.e. if the protection sum of the loader is invalid, this function returns 0. The protection code should not have any modifications, but if a cracker attempts to remove the protection or just to analyse it, this function fails and returns a zero value.

Return Value

If the protection is OK and has not been modified, the function returns 1, otherwise - if protection is corrupted - the return value is 0.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_CheckupIsEnigmaOk();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

See function examples in the installation folder, Examples\CheckEnigma subfolder.

EP_CheckupIsProtected

The EP_CheckupIsProtected function has a very simple purpose, it is used to check if the file is protected or not.

Return Value

The function returns 0 if the file is not protected and 1 if the file is protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_CheckupIsProtected();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

See function examples in the installation folder, Examples\CheckEnigma subfolder.

EP_EnigmaVersion

EP_EnigmaVersion returns the version of Enigma Protector the file is protected with.

Return Value

The return value contains the major version number in the high-order byte of the word and minor version number in the low-order byte. For example, return value 0x014E (334) means version 1.78.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_EnigmaVersion();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\CheckEnigma subfolder.

EP_MiscCountryCode

The EP_MiscCountryCode function returns the code of the current user's country. This value could be used for generating registration keys, see [Creating Keys](#).

Name	Code	Value
Afghanistan	CN_AFGHANISTAN	114
Albania	CN_ALBANIA	1
Algeria	CN_ALGERIA	2
Argentina	CN_ARGENTINA	3
Armenia	CN_ARMENIA	4
Australia	CN_AUSTRALIA	5
Austria	CN_AUSTRIA	6
Azerbaijan	CN_AZERBAIJAN	7
Bahrain	CN_BAHRAIN	8
Bangladesh	CN_BANGLADESH	115
Belarus	CN_BELARUS	9
Belgium	CN_BELGIUM	10
Belize	CN_BELIZE	11
Bolivia	CN_BOLIVIA	116
Bosnia and Herzegovina	CN_BOSNIA	117
Brazil	CN_BRAZIL	13
Brunei Darussalam	CN_BRUNEI	14
Bulgaria	CN_BULGARIA	15
Cambodia	CN_CAMBODIA	16
Canada	CN_CANADA	17
Caribbean	CN_CARRIBEAN	118
Chile	CN_CHILE	20
China	CN_CHINA	21
Colombia	CN_COLOMBIA	22
Costa Rica	CN_COSTARICA	23
Croatia	CN_CROATIA	24
Czech Republic	CN_CZECH	25
Denmark	CN_DENMARK	26
Dominican Republic	CN_DOMINICAN	27
Ecuador	CN_ECUADOR	28
Egypt	CN_EGYPT	29
El Salvador	CN_ELSALVADOR	30
Estonia	CN_ESTONIA	31
Ethiopia	CN_ETHIOPIA	119
Faroe Islands	CN_FAROE	32
Finland	CN_FINLAND	33
France	CN_FRANCE	34
Georgia	CN_GEORGIA	35
Germany	CN_GERMANY	36
Greece	CN_GREECE	37
Greenland	CN_GREENLAND	120
Guatemala	CN_GUATEMALA	38
Honduras	CN_HONDURAS	39
Hong Kong	CN_HONGKONG	40
Hungary	CN_HUNGARU	41
Iceland	CN_ICELAND	42
India	CN_INDIA	43
Indonesia	CN_INDONESIA	44
Iran	CN_IRAN	45
Iraq	CN_IRAQ	46
Ireland	CN_IRELAND	47
Israel	CN_ISRAEL	48
Italy	CN_ITALY	49
Jamaica	CN_JAMAICA	50
Japan	CN_JAPAN	51
Jordan	CN_JORDAN	52

Kazakhstan	CN_KAZAKHSTAN	53
Kenya	CN_KENYA	54
Korea	CN_KOREA	56
Kuwait	CN_KUWAIT	57
Kyrgyzstan	CN_KYRGYZSTAN	58
Laos	CN_LAOS	121
Latvia	CN_LATVIA	59
Lebanon	CN_LEBANON	60
Libyan	CN_LIBYAN	122
Liechtenstein	CN_LIECHTENSTEIN	62
Lithuania	CN_LITHUANIA	63
Luxembourg	CN_LUXEMBOURG	64
Macao	CN_MACAO	65
Macedonia	CN_MACEDONIA	66
Malaysia	CN_MALAYSIA	67
Maldives	CN_MALDIVES	123
Malta	CN_MALTA	124
Mexico	CN_MEXOCI	68
Monaco	CN_MONACO	70
Mongolia	CN_MONGOLIA	71
Montenegro	CN_MONTENEGRO	125
Morocco	CN_MOROCCO	72
Nepal	CN_NEPAL	126
Netherlands	CN_NETHERLANDS	73
New Zealand	CN_NEWZEALAND	74
Nicaragua	CN_NICARAGUA	75
Nigeria	CN_NIGERIA	127
Norway	CN_NORWAY	76
Oman	CN_OMAN	77
Pakistan	CN_PAKISTAN	78
Panama	CN_PANAMA	79
Paraguay	CN_PARAGUAY	80
Peru	CN_PERY	81
Philippines	CN_PHILIPPINES	82
Poland	CN_POLAND	83
Portugal	CN_PORTUGAL	84
Puerto Rico	CN_PUERTORICO	85
Qatar	CN_QATAR	86
Romania	CN_ROMANIA	87
Russia	CN_RUSSIA	88
Rwanda	CN_RWANDA	128
Saudi Arabia	CN_SAUDIARABIA	89
Senegal	CN_SENEGAL	129
Serbia	CN_SERBIA	130
Serbia and Montenegro	CN_SERBIAMONTENEGRO	90
Singapore	CN_SINGAROPE	91
Slovakia	CN_SLOVAKIA	92
Slovenia	CN_SLOVENIA	93
South Africa	CN_SOUTHAFRICA	94
Spain	CN_SPAIN	95
Sri Lanka	CN_SRILANKA	131
Sweden	CN_SWEDEN	96
Switzerland	CN_SWITZERLAND	97
Syrian	CN_SYRIAN	132
Taiwan	CN_TAIWAN	98
Tajikistan	CN_TAJIKISTAN	99
Thailand	CN_THAILAND	100
Trinidad and Tobago	CN_TRINIDADTOBAGO	101
Tunisia	CN_TUNISIA	102
Turkey	CN_TURKEY	103
Turkmenistan	CN_TURKMENISTAN	133
Ukraine	CN_UKRAINE	104
United Arab Emirates	CN_UAE	105

United Kingdom	CN_UNITEDKINGDOM	106
United States	CN_USA	107
Uruguay	CN_URUGUAY	108
Uzbekistan	CN_UZBEKISTAN	109
Venezuela	CN_VENEZUELA	110
Viet Nam	CN_VIETNAM	111
Yemen	CN_YEMEN	112
Zimbabwe	CN_ZIMBABWE	113

Parameters

The function does not have parameters.

Return Value

If the function succeeds, the return value is the code of the user's country, otherwise it returns 0.

Remark

The function fails in the following cases:

- the application is not protected.

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_MiscCountryCode();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples subfolder.

EP_MiscGetWatermark

The EP_MiscGetWatermark function returns watermarks contents. To learn more about watermarks, refer to [Miscellaneous - Watermark](#).

Parameters

- | **ID** - the number of the watermark to return;
- | **WM** - a pointer to TWMCContent struct. If this parameter is NULL, the function returns the count of watermarks;
- | **WM.WMType** - returns the watermark type;
- | **WM.Name** - a pointer to the buffer to return Name of the watermark to. If it is NULL, NameLen returns the required buffer length;
- | **WM.NameLen** - the size of the buffer for the watermark Name;
- | **WM.Text** - a pointer to the buffer to return Text of the watermark to. If it is NULL, TextLen returns the required buffer length;
- | **WM.TextLen** - the size of the buffer for the watermark Text;
- | **WM.FileName** - a pointer to the buffer to return FileName of the watermark to. If it is NULL, FileNameLen returns the required buffer length;
- | **WM.FileNameLen** - the size of the buffer for the watermark FileName;
- | **WM.AFile** - a pointer to the buffer to return File of the watermark to. If it is NULL, AFileLen returns the required buffer length;
- | **WM.AFileLen** - the size of the buffer for the watermark File;

Return Value

If the function succeeds, the return value is the count of watermarks.

If Name parameter of TWMCContent is NULL or the length of the buffer that is placed into NameLen variable is less than required, the function returns the necessary buffer length in the NameLen variable.

If Text parameter of TWMCContent is NULL or the length of the buffer that is placed into TextLen variable is less than required, the function returns the necessary buffer length in the TextLen variable.

If FileName parameter of TWMCContent is NULL or the length of the buffer that is placed into FileNameLen variable is less than required, the function returns the necessary buffer length in the FileNameLen variable.

If AFile parameter of TWMCContent is NULL or the length of the buffer that is placed into AFileLen variable is less than required, the function returns the necessary buffer length in the AFileLen variable.

If the function fails, the return value is 0.

Remark

The function fails in the following cases:

- | no watermarks in the protected file;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllexport ) __stdcall int EP_MiscGetWatermark( int ID, PWMCContent WM );
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

See function examples in the installation folder, Examples\Watermarks subfolder.

EP_ProtectedStringByID

The EP_ProtectedStringByID function returns protected strings. See also [Protection Features - Protected Strings](#).

Parameters

- | **ID** - the ID of the protected string - an integer value.
- | **Buffer** - a pointer to the buffer for the protected string. If this parameter is NULL, the function returns the necessary size of buffer.
- | **Len** - the size of the Buffer for the protected string.

Return Value

The function returns the size of the buffer for a protected string or 0 if the function fails.

Remark

The function fails in the following cases:

- | there is no protected string with the specified ID;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_ProtectedStringByID( int Total, char* Left, int Len);
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

[Show/Hide C# \(.NET\) function example](#)

[Show/Hide Visual Basic function example](#)

See function examples in the installation folder, Examples\ProtectedStrings subfolder.

EP_ProtectedStringByKey

The EP_ProtectedStringByKey function returns protected strings. See also [Protection Features - Protected Strings](#).

Parameters

- | **Key** - a pointer to the null terminated string that contains a key of the protected string.
- | **Buffer** - a pointer to the buffer for the protected string. If this parameter is NULL, the function returns the necessary size of the buffer.
- | **Len** - the size of the buffer for the protected string.

Return Value

The function returns the size of the buffer for a protected string or 0 if the function fails.

Remark

The function fails in the following cases:

- | there is no protected string with the specified Key;
- | the application is not protected.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_ProtectedStringByKey( char* Key, char* Left, int Len);
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

[Show/Hide C# \(.NET\) function example](#)

[Show/Hide Visual Basic function example](#)

See function examples in the installation folder, Examples\ProtectedStrings subfolder.

EP_SplashScreenShow

The EP_SplashScreenShow function forces showing of a splash screen. If the Splash Screen is already shown, the function does nothing. It returns a handle of the Splash Screen window, you may place your own controls onto this window. Splash Screen feature should be enabled, otherwise the function will fail, see [MISCELLANEOUS - Splash Screen](#).

Return Value

The function returns the handle of the splash screen window, or zero if it fails.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall int EP_SplashScreenShow();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

See examples in the Examples\SplashScreen folder.

EP_SplashScreenHide

The EP_SplashScreenHide function hides the Splash Screen (if it is shown). See [MISCELLANEOUS - Splash Screen](#).

Definition

[\[-\] Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall void EP_SplashScreenHide();
```

[\[+\] Show/Hide Delphi function definition](#)

[\[+\] Show/Hide Visual Basic function definition](#)

[\[+\] Show/Hide C# \(.NET\) function definition](#)

Examples

[\[+\] Show/Hide Delphi function example](#)

See examples in the Examples\SplashScreen folder.

EP_CheckupVirtualizationTools

The `EP_CheckupVirtualizationTools` function allows checking if the file is started under any of the known Virtualization Tools like VMWare/VirtualBox/VirtualPC. Note that you have to enable the [CHECK-UP - Virtualization Tools](#) feature, select the necessary virtualization tools you would like to check and disable the Terminate Execution option.

Return Value

The function returns 1 (true) if the file is started under any virtualization tool or 0 (false) if the function fails. See the Remark section below for the cases of function failure.

Remark

The function fails in the following cases:

- ▮ the application is not protected;
- ▮ the [CHECK-UP - Virtualization Tools](#) feature is not enabled;
- ▮ necessary Virtualization Tools for checking have not been selected in the [CHECK-UP - Virtualization Tools](#) panel.

Definition

[Show/Hide C++ function definition](#)

```
extern "C" __declspec( dllimport ) __stdcall BOOL EP_CheckupVirtualizationTools();
```

[Show/Hide Delphi function definition](#)

[Show/Hide Visual Basic function definition](#)

[Show/Hide C# \(.NET\) function definition](#)

Examples

[Show/Hide Delphi function example](#)

See function examples in the installation folder, `Examples\CheckVirtualizationTools` subfolder.

How to Use Enigma API

Enigma Protector allows using Enigma API through the call of `enigma_ide.dll` functions from import table or through the pair of functions `GetModuleHandle/LoadLibrary` and `GetProcAddress`. Note, that Enigma API uses `stdcall` calling conversion. The common method of Enigma API usage (through importing Enigma API) can be found in the Enigma Protector installation folder, `Examples` subfolder. Here are examples demonstrating one more way to call Enigma API:

Delphi

```
var
  pEP_RegHardwareID : function : pchar;
  pEP_RegCheckKey : function(pcName, pcKey : pchar) : boolean;
  pcName : pchar;
  pcKey : pchar;

begin
  // Show message box with a Hardware ID
  pEP_RegHardwareID := GetProcAddress(GetModuleHandle('enigma_ide.dll'), 'EP_RegHardwareID');
  MessageBox(0, pEP_RegHardwareID, 'Application', MB_OK);
  // Check registration information
  pcName := 'Registration Info';
  pcKey := 'Registration Key';
  pEP_RegCheckKey := GetProcAddress(GetModuleHandle('enigma_ide.dll'), 'EP_RegCheckKey');
  if pEP_RegCheckKey(pcName, pcKey) then
    MessageBox(0, 'Valid Registration Key', 'Application', MB_OK)
  else
    MessageBox(0, 'Invalid Registration Key', 'Application', MB_OK)
end;
```

FAQ

In these sections you will find the answers to the most frequently asked questions.

- | [API FAQ](#) - questions about using The Enigma API.
- | [Keys Generator FAQ](#) - questions about using the built-in and customized generators of registration keys.
- | [Protector FAQ](#) - questions about the work of the protector features.
- | [Other FAQ](#) - all questions not related to the other topics.
- | [Registration Manager FAQ](#) - questions regarding Enigma Protector Registration Manager.
- | [Mailer FAQ](#) - questions regarding Enigma Protector Mailer.

API FAQ

Q: I'm a Visual Basic developer and I can't figure out why Enigma API failed after execution? Here is the part of my sources:

```
If EP_RegLoadAndCheckKey = TRUE Then
    ' *****
End If
```

The codes between If - End If have never been executed.

A: You have a mistake in the first line of the above source, the expression `EP_RegLoadAndCheckKey = TRUE` will never be executed in Visual Basic. It occurs due to the peculiarities of the Visual Basic language, where a TRUE value means \$FFFF, a FALSE value means 0, but Enigma API's return 0 if a function fails or 1 if it succeeds. Use the following scheme to solve your problem:

```
If EP_RegLoadAndCheckKey Then
    ' *****
End If
```

Q: Do I need to distribute `enigma_api.dll` with the protected module?

A: No, you don't, `enigma_api.dll` is just an empty library which exports empty Enigma API's. This library is used for debugging of a non-protected module in development, so this library is not needed after the module is protected.

Q: Is it possible to use Enigma API with Visual Basic .NET applications?

A: Yes, it is possible! Enigma API definitions for Visual Basic .NET are similar to the Visual Basic ones. See `Example\` folder for examples of API usage and `EnigmaSDK\VB\` for API definitions.

Keys Generator FAQ

Q: I have several users who are able to generate registration keys for our software. I need some service to provide simultaneous access to the keys database and to the keys generator by several users. How can I place a custom registration keys generator into a share folder?

A: We have a great idea about how to grant access to the keys base and keys generator by means of MS Access service. See the Custom Keys Generator help topic, KeyGen subtopic on the shared folder.

Q: I'm using a unicode registration scheme, how can I generate registration keys with unicode support?

A: After you have selected the unicode registration scheme check box in the Registration Features - Common panel, the Keys Generator (which is accessed through the Main Menu) and keys generator of License Manager automatically begin generating unicode registration keys. To generate registration keys with unicode support with CGI keys generator or keygen.dll, you have to use actions (for CGI keys generator) and functions (for keygen.dll) with a W prefix.

Q: I have a few applications protected with Enigma. All applications require a registration key to run. If I register one of the program with a registration key, other programs become registered too, but I want to register each application with its own registration key... What am I doing wrong?

A: If all applications accept one registration key, this means that all of them are protected with the same project file. To have a unique registration scheme for each application, you need to create a new project in Enigma for each protected program. Each new project has unique secure registration parameters that are used for generating registration keys, so after you create a new project, each program will be registered with its own registration key.

Other FAQ

Q: I need a special build of The Enigma Protector with extended options. Can you make it?

A: Yes, it is possible. But your offer should contain legality and safety rights! Please, contact our support team by email.

Protector FAQ

Q: I have lost The Enigma Protector project file and I can't protect my module again and can't generate registration keys for current module, how can I fix it?

A: Sorry, but this is impossible. The Enigma Protector project file contains several unique constants that are used for key generation. These constants are generated when a project is created and have random contents, and therefore we can't restore them! Please, make backup copies of your project files!

Q: I have several versions of my software, one is protected with The Enigma Protector, the other is not. How can I check whether the current version is protected?

A: This can be solved by using a [Reg_Crypt](#) macros. See the following Delphi example:

```
function IsEnigmaPresent : boolean;
begin
  Result := true;
  // this code returns TRUE at any ways,
  // but we should use it to fighting
  // with the Delphi code optimization
  if GetModuleHandle(nil) <> 0 then
  begin
    {$I reg_crypt_begin1.inc}
    Result := false;
    {$I reg_crypt_end1.inc}
  end;
end;
```

Registration Manager FAQ

Q: Where does Enigma Protector store the registrations database?

A: Enigma Protector stores the database (registrations & customers information) in a file with the same name as the project file but with a .enigmadb extension.

Mailer FAQ

Q: Where does Enigma Protector store Mailer settings/database?

A: Enigma Protector stores Mailer settings and emails database in a file with the same name as the project file but with .enigmamail extension.

Q: I can't send emails with my gmail account, Mailer reports error...?

A: Gmail does not allow insecure connection (which is used by default), to solve the problem just enable the SSL option in mailer properties.

Getting Started

The most important thing in module protection is the correctness of protection scheme building. Even the most sophisticated protection options can be easily eliminated by crackers in case you made an error while applying the protection. There are several pieces of advice one has to follow to make module protection really difficult for cracking:

- | if you are using registration keys features, it is strongly recommended to use crypt macros to limit the functionality of a module. The crypt macros are the highest protection feature, the code between the macros is deciphered only when the module is registered. The unique key data for deciphering are placed only in the registration key, so it is impossible to crack the module without a registration key.
- | the common mistake of software developers is using trial features without any trial limitations. Do not make a trial application fully functional. There are no hundred-per-cent methods to make trial limitations uncrackable, the trial can only provide much more time to make your application stable against cracker attacks.
- | only demonstration versions can fully secure you against crackers. A demonstration version contains only basic functions and has no any special features. These versions can't be cracked because these just don't contain necessary sources.

Let's go!

To create a protection scheme for the module, follow these steps:

1. [Create a unique Enigma project.](#)
2. [Select necessary input parameters.](#)
3. [Select protection parameters.](#)
4. (Optional) [Select trial parameters.](#)
5. (Optional) [Select registration key parameters.](#)
6. (Optional) Modification of the module source code.
7. Module protection.

Choosing input parameters

Input parameters are the most important thing needed to start the protection, here are the basic parameters:

1. the file name to protect, make sure that the file is supported by Enigma Protector
2. the name and version of the product (module), this information must be unique for every project since it is used to create some unique protection parameters
3. the file name of the protected module.

Choosing protection parameters

Choose the necessary protection parameters.

- | [Check-Up](#)
- | [Protection Features](#)
- | [Obfuscation](#)
- | [Miscellaneous](#)

Creating a protection project

Creating a protection project is the most important part in building the protection security for the protected module. A project file has a set of protection parameters and several unique constants which are used for generation and verification of registration keys. Unique constants are generated randomly every time a project is created and their coincidence for several projects is impossible.

Warnings:

- | Do not use project files distributed with the example programs of the Enigma Protector! These files can be used by anyone, so anyone will be able to generate registration keys for your project.
- | Make a backup copy of the existing project file! In case the project file is lost, you will not be able to restore the unique data and generate registration keys for the existing project!

Registration keys features

Choosing parameters used for registration keys features. The Enigma Protector supports options to generate registration keys for protected modules that will allow you to easily convert software to shareware and distribute registration keys to customers. The registration keys create dependencies on user information, user's hardware (optional), key creation date, key expiration date (optional), names of crypted sections (optional). These features provide a flexible system to create protection of the existing modules with limited usage. A dissymmetric algorithm of enciphering identical to RSA algorithm with a 512-bit length is used to create registration keys. It provides an exceptionally high level of security for the protected module, preventing the reconstruction or reversing of a registration key. It is necessary to define the parameters of registration information storage and type of hardware locking before module protection (See [Program Overview](#))

The registration of the protected module can be carried out in the following ways:

1. Use of files with the registration information. Subject to the selected registration info storage option, it can be sent to user as a reg file (the user will need to add data from the reg file to the Windows registry by running the file) or as a key file (the file with the registration information will have to be copied into the folder specified in the Registration data storing panel).
2. Use of internal Enigma API functions for checking, reading, saving, deleting the registration information. In this case you should make respective alterations in the source code of the module, provide dialog windows for entering the registration information, provide the registration information for checking and saving.

Trial limitations

Choosing trial period parameters. This feature serves for limiting the usage period of an unregistered version of the protected module on the user's PC. The Enigma Protector supports the following types of trial limitations: by count of module executions, by count of usage days, by expiration date. All trial limitations can be controlled manually by using Enigma API, in this case you can fully control the trial counter and make a handler for any changes (for example, you can warn a user to register the protected module several days/executions before the trial expiration). In case you don't use Enigma API, you can define the type of a trial expiration event manually in the TRIAL CONTROL panel. Also, The Enigma Protector supports reminder messages (nag-screens) for the protected module, the reminder message is intermittently displayed during the work of the module with a predefined time interval. Is not a secret that some users may set the system clock backward to restore the functionality of the protected module - to prevent this, Enigma has a feature to detect system clock reversing. Rest assured, the system clock checkup is not related to the trial limitation so it will not be disabled after the module registration

Warning: all the above trial limitations will be lifted after the protected module is registered. Refer to [Program Overview](#) for a detailed description of trial features.

Keys generator

This section explains how to use the key generator. Use the following links to get detailed information:

- | [Creating Keys](#)
- | [Key Verification](#)
- | [Custom Key Generator](#)
- | [Export Key Generator](#)
- | [License Manager](#)
- | [CGI Key Generator \(Web based\)](#)

Creating Keys

To create a registration key for the existing project you can use special Enigma Protector service - registration key generator. The generator of registration keys can be built into the Enigma Protector or user shell.

Using the built-in registration key generator.

Follow these steps to use the built-in keys generator:

- 1 choose the output registration key type in the [Registration Features - Common](#) panel.
- 2 open The Enigma Protector and choose Main menu-Tools-Keys Generator menu.

Enigma Key generator (UNICODE)

Registration name:

Registration key:

Add hyphens

Copy

Paste

Verify

Result of the verification:

Expiration Date 1/12/2010

Register After 1/12/2010

Register Before 1/12/2010

Country Lock Albania

Hardware ID

Executions 100

Days 30

Run Time 10 mins

Global Time 60 mins

Select sections for decryption

Section 1 Section 5 Section 9 Section 13

Section 2 Section 6 Section 10 Section 14

Section 3 Section 7 Section 11 Section 15

Section 4 Section 8 Section 12 Section 16

Generate Close

- 1 Registration name - specific information that will define the user that the registration key is generated for. The registration name should be distributed together with the registration key. This field can't be empty. Please note, the registration information can be used by the key generator in the ANSI or unicode format, select the type of registration scheme in the [Registration Features - Common](#) panel. The current type of registration scheme (ANSI or UNICODE) is shown in the window header;
- 2 Add hyphens - the key will contain hyphens;
- 3 Expiration date - the date after which the registration key will become invalid. To view the expiration date of the current registration key inside your application, use the Enigma API function [EP_ReqKeyExpirationDate](#);

- | Register After - set this date to limit the start date of the allowed registration period. If the registration is performed before this day, the key will be deemed invalid, otherwise the key will be valid. To return the Register After date inside your application use the Enigma API [EP_RegKeyRegisterAfterDate](#);
- | Register Before - set this date to limit the end date of the allowed registration period. If the registration is performed before this day, then the key will be deemed valid, otherwise the key will be invalid. To return the Register Before date inside your application use the Enigma API [EP_RegKeyRegisterBeforeDate](#);
- | Country - select the country the generated key will be locked to. This feature means that the registration will only be valid on a PC located in the specified country. To get the country of the user PC use the Enigma API [EP_MiscCountryCode](#);
- | Executions - enter the number of executions to limit the registration key to. The registration key will only be valid for the specified number of executions of the unregistered protected application. To get information about the total and remaining number executions inside your application use the Enigma API [EP_RegKeyExecutions](#);
- | Days - enter the number of days to limit the registration key to. The registration key will only be valid for the specified number of day since the first execution of the protected application. To get the total and remaining number of days use the Enigma API [EP_RegKeyDays](#);
- | Run Time - enter the number of minutes the registration key will be valid for since the protected application is run. The registration key will become valid at each execution and expire after the specified number of minutes. To get the number of remaining and total Run-time minutes inside your application use the Enigma API [EP_RegKeyRuntime](#);
- | Global Time - enter the number of minutes of Global Time to limit the registration key to. The Global Time means the sum of the minutes of all executions of the registered protected application. To get the number of remaining and total Global Time minutes while the application is working use the Enigma API [EP_RegKeyGlobalTime](#);
- | Hardware ID - the unique computer identifier. This feature allows locking the registration key to a particular computer, i.e. the registration key will only be valid on the computer having the hardware ID used for key generation. Use the Enigma API function [EP_RegHardwareID](#) to get the hardware ID string from the user's PC. See the [Hardware Lock](#) panel for the options available for hardware locking.
- | Section - select the encrypted section that will be decrypted with the generated registration key. See [Markers Reg_Crypt](#) to learn more about encrypted sections.

Using custom registration key generators.

Enigma Protector supports a service for creating custom shells of registration key generators (See [Custom Keys Generator](#)).

Key Verification

The verification of registration keys is performed in the same window as registration key creation (See [Creating Keys](#)). Fill out the "Registration name" and "Registration key" fields (if the key is hardware locked then "Hardware ID" has to be entered too) and click the "Verify" button to validate the key. In the "Result of the verification" field you will see the results. Also, take a look at the [Custom Keys Generator](#) with keygen libraries - there are a few functions that allow verifying registration info in your own key generator.

Example:

Registration name :

Vladimir

Registration key :

W3UH9K-L5EUT2-XUPUYJ-P8YYHR

Verification results :

Valid registration key.
Key information:
Created: 17 April 2007
Expiration date: 1 January 2008
Unlock crypted section #1
Unlock crypted section #7
Unlock crypted section #16

It means the key is valid for the current project and contains the following information:

- | key creation date: 17 April 2007;
- | key expiration date: 1 January 2008;
- | key deciphers the following crypted sections #1, #7, #16.

CGI Key Generator

CGI Key Generator can be used for generating registration keys for certain projects through the remote server call, web page, web request. CGI key generator should be placed on the server (there are 2 kinds of CGI key generators, for Windows and for Linux servers, you can find it in EnigmaSDK\CGI Keygen folder) into the cgi-bin folder, and the execution attributes (permissions) for the key generator file should be enabled. Calling the CGI generator with the necessary parameters will generate a registration key. The key generator parameters could be passed to the CGI key generator by GET or POST method. Remote CGI key generator is the best way to integrate a key generator with automatic registrars like ShareIt, RegNow, Plimus etc.

How to use the CGI key generator:

1. Place the keygen to the cgi-bin folder of your web site (make sure that your server supports execution of CGI applications)
2. Enable execution permissions for the keygen file ("keygen" for Linux and "keygen.exe" for Windows servers)
3. Create a link for key generator execution:

There are the following actions that could be used with CGI key generator:

- | GenerateKey - generates ANSI* registration keys;
- | GenerateKeyA - the duplicate of GenerateKey;
- | GenerateKeyW - generates unicode* (wide string) registration keys;
- | GenerateKeyFromProject - generates ANSI* registration keys, generator parameters** will be extracted from the project file;
- | GenerateKeyFromProjectA - the duplicate of GenerateKeyFromProject;
- | GenerateKeyFromProjectW - generates unicode* (wide string) registration keys, generator parameters** will be extracted from the project file.

* please note that you should only use ANSI or unicode scheme of licensing for a particular project. If the [UNICODE Registration Scheme](#) option is enabled in the [REGISTRATION FEATURE - Common](#) panel, the key generator should be called with a unicode support action (actions with W prefix), otherwise ANSI actions should be used.

** generator parameters - KeyMode, KeyBase, EncryptedConstant, PublicKey, PrivateKey.

CGI Key Generator accepts the following parameters:

Action - defines the action to be performed, allows the following values:

- | GenerateKey
- | GenerateKeyA
- | GenerateKeyW
- | GenerateKeyFromProject
- | GenerateKeyFromProjectA
- | GenerateKeyFromProjectW

FileName - the name of the project file. This parameter is used only for actions that use the project file (like GenerateKeyFromProject);

RegName - enter a registration name for the registration key. Please note that this string should be http encoded to avoid appearing of untranslated symbols (for example, the space symbol should be translated into %20 but not " ");

KeyMode - defines what type of registration key should be generated. To get this value, open your project file in Enigma and go to REGISTRATION FEATURES-Common panel, take a look at the Registration key safety/length field, for example, if this field is RSA 1024, you should use KeyMode=1024 (possible values for KeyMode are 512/768/1024/2048/3072/4096);

KeyBase - defines the output format of a registration key. To get this value, open your project file in Enigma and go to REGISTRATION FEATURES-Common panel, take a look at the Registration key output base field, for example, if this field is Base 32, then you should use KeyBase=32 (possible values for KeyBase are 2/8/16/32/64);

Hyphens - indicates if the registration key should be divided by hyphens or not. If yes, enter Hyphens=1, otherwise skip this parameter;

Hardware - the hardware ID the registration key should be locked to;

Expiration - the registration key expiration date, define this parameter with the expiration date value*;

RegAfter - Register After date*;

RegBefore - Register Before date*;

Executions - the number of executions to limit the registration key to;

Days - the number of days to limit the registration key to;

Runtime - the number of run-time minutes to limit the registration key to;

Globaltime - the overall number of minutes to limit the registration key to;

Country - the code of the county to lock the registration key to;

Sections - if the registration key should unlock crypted sections, generate a 16-digit string indicating the section that should be unlocked with the generated key. If digit = 1, the key will unlock a section, or 0 if no section should be unlocked. The section string 1010001001001001 shows that sections #1, #3, #7, #10, #13, #16 should be unlocked

EncryptedConstant - get this constant from the project file, open project file in notepad and find there EnigmaProject-RegistrationFeatures-Constants-EncryptedConstant branch and get its value

PrivateKey and *PublicKey* - values should be looked up in the project file. To get these values, open your project file in Enigma and go to the [REGISTRATION FEATURES - Common](#) panel, the Public and Private Keys are shown in the "Information for Custom Keys Generator" box.

* the date value should be represented in a string. The date has the following format: 2 digits (day) + 2 digits (month) + 4 digits (year). For example, 1 Dec 2010 date is string 01122010

Examples

[⊕ Show/Hide HTML \(GET method\) example](#)

[⊕ Show/Hide HTML \(GET method\) getting parameters from project file example](#)

[⊕ Show/Hide HTML \(POST method\) example](#)

[⊕ Show/Hide HTML \(POST method\) getting parameters from project file example](#)

[⊕ Show/Hide PHP \(POST method\) example](#)

[⊕ Show/Hide PHP \(GET method\) example](#)

We recommend using the POST method to pass data to the key generator, because the GET method has a data size limit that can be passed through the url string.

Export Key Generator

Enigma Protector allows exporting the key generator for the existing project into a single execution file. This option may be useful if you do not want to run/use Enigma itself and the project file to generate licenses. Simple exported key generator has a user-friendly interface, it does not require any external files (like the project file) and all project data are already embedded into the exe file. To export the key generator click Main Menu - Export Key Generator. Once you have clicked that, follow a few steps:

Step 1. Default value for generated keys. Should a registration key be hyphenated? If you do not want to allow users of this key generator to change the hyphens value, enable the "Do not allow changing hyphens value" option.



Step 2. Default value for unlocked sections. If you do not want to allow users of this key generator to change the sections value, enable the "Do not allow changing sections value" option.



Step 3. Default value for expiration date of generated keys. If you do not want to allow users of this key generator

to change the expiration date value, enable the "Do not allow changing expiration date value" option.



Step 4. Default value for Register After date. If you do not want to allow users of this key generator to change the Register After date value, enable the "Do not allow changing Register After date" option.



Step 5. Default value for Register Before date. If you do not want to allow users of this key generator to change the Register Before date value, enable the "Do not allow changing Register Before date" option.



Step 6. Default value for executions limit. If you do not want to allow users of this key generator to change the executions value, enable the "Do not allow changing Executions " option.



Step 7. Default value for days limit. If you do not want to allow users of this keys generator to change the days value, enable the "Do not allow changing " option.



Step 8. Default value for run-time limit. If you do not want to allow users of this key generator to change the run-time value, enable the "Do not allow changing Run-time " option.



Step 9. Default value for global time limit. If you do not want to allow users of this keys generator to change the global time value, enable the "Do not allow changing Global Time " option.



Step 10. Default value for country lock. If you do not want to allow users of this keys generator to change the country lock value, enable the "Do not allow changing Country Lock " option.



Step 11. Click the Finish button and enter the file name to export the keygen. Try it!

Custom Key Generator

Enigma Protector provides services for custom generation of registration keys for your own projects. There are special libraries (written in Windows and Linux versions) `keygen.dll` (for Windows) and `libkeygen.so` (For Linux) that allow generating and verifying registration keys. Enigma Protector provides freedom in development of the custom shells for key generators, examples of such custom key generators written for different compilers can be found in `Examples\Keygen` folder and `Examples\KeygenUnicode` for projects using a unicode registration scheme. The libraries export the following functions:

For ANSI-based registration information:

- | `KG_GenerateRegistrationKey`
- | `KG_GenerateRegistrationKeyFromProject`
- | `KG_GenerateRegistrationKeyA`
- | `KG_GenerateRegistrationKeyFromProjectA`
- | `KG_VerifyRegistrationInfo`
- | `KG_VerifyRegistrationInfoFromProject`
- | `KG_VerifyRegistrationInfoA`
- | `KG_VerifyRegistrationInfoFromProjectA`

For unicode-based registration information:

- | `KG_GenerateRegistrationKeyW`
- | `KG_GenerateRegistrationKeyFromProjectW`
- | `KG_VerifyRegistrationInfoW`
- | `KG_VerifyRegistrationInfoFromProjectW`

TKeyGenParams type

TKeyGenParams structure is used by KG_GenerateRegistrationKey and KG_GenerateRegistrationKeyFromProject to generate registration keys.

Structure description

KeyMode	(input parameter) mode of registration key (RSA 512/768/1024/2048/3072/4096). It may have the following values (see Registration Features - Common Registration key safety/length value): RM_512 = 0; RM_768 = 1; RM_1024 = 2; RM_2048 = 3; RM_3072 = 4; RM_4096 = 5;
KeyBase	(input parameter) base of registration key (Base 2/8/16/32/64). It may have the following values (see Registration Features - Common Registration key output base value): RB_2 = 0; RB_8 = 1; RB_16 = 2; RB_32 = 3; RB_64 = 4;
KeyWithHyphens	(input parameter) add hyphens to the key, set to 1 if true or 0 if false (see Creating Keys);
Key	(input parameter) pointer to the memory where the registration key will be placed. The registration key will be represented as ANSI, null terminated string. Please be sure that the buffer allocated for the registration key has sufficient size, we recommend allocating a buffer at least 2048 bytes long;
KeyLen	(input parameter) size of the registration key buffer;
RegInfo	(input parameter) pointer to the memory where the registration name info is placed ;
RegInfoLen	(input parameter) size of registration info;
UseKeyExpiration	(input parameter) if the value is 0, then the key does not have expiration, if the value is 1 then the key is time-limited. The key will expire on the date specified in the ExpirationYear, ExpirationMonth, ExpirationDay parameters (see Creating Keys);
ExpirationYear	(input parameter) year of key expiration date;
ExpirationMonth	(input parameter) month of key expiration date;
ExpirationDay	(input parameter) day of key expiration date;
UseHardwareLocking	(input parameter) if the key is hardware-locked, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
HardwareIDType	(input parameter) pointer to ANSI, null terminated hardware id string;
UseExecutionsLimit	(input parameter) to enable executions limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
ExecutionsCount	(input parameter) number of executions to limit the registration key to;
UseDaysLimit	(input parameter) to enable days limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
DaysCount	(input parameter) number of days to limit the registration key to;
UseRunTimeLimit	(input parameter) to enable run-time limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);

RunTimeMinutes	(input parameter) number of run-time minutes to limit the registration key to;
UseGlobalTimeLimit	(input parameter) to enable global time limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
GlobalTimeMinutes	(input parameter) number of global time minutes to limit the registration key to;
UseCountyLimit	(input parameter) to enable country limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
CountryCode	(input parameter) the country code to lock the registration key to (see Enigma API EP_MiscCountryCode to get the code of a particular country);
UseRegisterAfter	(input parameter) to enable the Register After limit for the registration key, set this value to 1, otherwise it should be 0. The Register After date is specified in the RegisterAfterYear, RegisterAfterMonth, RegisterAfterDay parameters (see Creating Keys);
RegisterAfterYear	(input parameter) year of Register After date;
RegisterAfterMonth	(input parameter) month of Register After date;
RegisterAfterDay	(input parameter) day of Register After date;
UseRegisterBefore	(input parameter) to enable the Register Before limit for the registration key, set this value to 1, otherwise it should be 0. The Register Before date is specified in the RegisterBeforeYear, RegisterBeforeMonth, RegisterBeforeDay parameters (see Creating Keys);
RegisterBeforeYear	(input parameter) year of Register Before date;
RegisterBeforeMonth	(input parameter) month of Register Before date;
RegisterBeforeDay	(input parameter) day of Register Before date;
EncryptedConstant	(input parameter) integer constant, obtain it from the project file (see Registration Features - Common Registration, Information for Custom Keys Generator box, Encryption Constant value);
EncryptedSections	(input parameter) array of 16-byte length, which points to the encrypted sections that will be decrypted by this key (see Creating Keys);
PublicKey	(input parameter) pointer to ANSI, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Public Key value);
PrivateKey	(input parameter) pointer to ANSI, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Private Key value).

Definition

- [⊕ Show/Hide Delphi structure definition](#)
- [⊕ Show/Hide C++ structure definition](#)
- [⊕ Show/Hide C# \(.NET\) structure definition](#)
- [⊕ Show/Hide Visual Basic structure definition](#)

Function KG_GenerateRegistrationKey

The function is used to generate ANSI registration keys. Note: to enable the application to accept ANSI registration keys, the UNICODE Registration Scheme parameter from the [Registration Features - Common](#) panel should be disabled. The function has one parameter - PKeyGenParams, which is a pointer to the [TGenKeyParams](#) structure. This function requires the following necessary input parameters: PublicKey, PrivateKey, KeyMode, KeyBase and EncryptedConstant (other parameters are optional) that should be obtained from the project (see the [Registration Features - Common](#) panel, Information for Custom Generator box).

Return Values

EP_NO_ERROR=0	the function succeeds. In this case, Key parameter of TKeyGenParams structure should contain a pointer to the null terminated string - registration key.
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

- [+ Show/Hide Delphi function definition](#)
- [+ Show/Hide C++ function definition](#)
- [+ Show/Hide C# \(.NET\) function definition](#)
- [+ Show/Hide Visual Basic function definition](#)

Examples

- [+ Show/Hide Delphi function example](#)
- [+ Show/Hide Borland C++ Builder function example](#)
- [+ Show/Hide Visual C++ function example](#)
- [+ Show/Hide C# \(.NET\) function example](#)
- [+ Show/Hide Visual Basic function example](#)

See function examples in the installation folder, Examples\Keygen subfolder.

Function KG_GenerateRegistrationKeyFromProject

The function is similar to [KG_GenerateRegistrationKey](#). The main difference of this function is that it reads secure information (PublicKey, PrivateKey, KeyMode, KeyBase and EncryptedConstant) from the project file and you do not have to set up these parameters in the input structure [TGenKeyParam](#). [KG_GenerateRegistrationKeyFromProject](#) has one additional parameter - ProjectFile. ProjectFile is a pointer to ANSI - null terminated string, which defines the path to the Enigma Protector project file.

Warning: ProjectFile has to contain an absolute path to the project file.

Return Values

EP_NO_ERROR=0	the function succeeds. In this case, Key parameter of TKeyGenParams structure should contain a pointer to the null terminated string - registration key.
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

[+ Show/Hide Visual Basic function definition](#)

Examples

[+ Show/Hide Delphi function example](#)

[+ Show/Hide Borland C++ Builder function example](#)

[+ Show/Hide Visual C++ function example](#)

[+ Show/Hide C# \(.NET\) function example](#)

[+ Show/Hide Visual Basic function example](#)

See function examples in the installation folder, Examples\Keygen subfolder.

TKeyGenParamsA type

TKeyGenParamsA structure is used by KG_GenerateRegistrationKeyA and KG_GenerateRegistrationKeyFromProjectA to generate registration keys.

Structure description

KeyMode	(input parameter) mode of registration key (RSA 512/768/1024/2048/3072/4096). It may have the following values (see Registration Features - Common Registration key safety/length value): RM_512 = 0; RM_768 = 1; RM_1024 = 2; RM_2048 = 3; RM_3072 = 4; RM_4096 = 5;
KeyBase	(input parameter) base of registration key (Base 2/8/16/32/64). It may have the following values (see Registration Features - Common Registration key output base value): RB_2 = 0; RB_8 = 1; RB_16 = 2; RB_32 = 3; RB_64 = 4;
KeyWithHyphens	(input parameter) add hyphens to the key, set to 1 if true or 0 if false (see Creating Keys);
Key	(input parameter) pointer to the memory where the registration key will be placed. The registration key will be represented as ANSI, null terminated string. Please be sure that the buffer allocated for the registration key has sufficient size, we recommend allocating a buffer at least 2048 bytes long;
KeyLen	(input parameter) size of the registration key buffer;
RegInfo	(input parameter) pointer to ANSI, null terminated string, which is the registration name;
UseKeyExpiration	(input parameter) if the value is 0, then the key does not have expiration, if the value is 1, then the key is time-limited. The key will expire on the date specified in the ExpirationYear, ExpirationMonth, ExpirationDay parameters (see Creating Keys);
ExpirationYear	(input parameter) year of key expiration date;
ExpirationMonth	(input parameter) month of key expiration date;
ExpirationDay	(input parameter) day of key expiration date;
UseHardwareLocking	(input parameter) if the key is hardware-locked, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
HardwareIDType	(input parameter) pointer to ANSI, null terminated hardware id string;
UseExecutionsLimit	(input parameter) to enable executions limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
ExecutionsCount	(input parameter) number of executions to limit the registration key to;
UseDaysLimit	(input parameter) to enable days limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
DaysCount	(input parameter) number of days to limit the registration key to;
UseRunTimeLimit	(input parameter) to enable run-time limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
RunTimeMinutes	(input parameter) number of run-time minutes to limit the registration key to;

UseGlobalTimeLimit	(input parameter) to enable global time limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
GlobalTimeMinutes	(input parameter) number of global time minutes to limit the registration key to;
UseCountyLimit	(input parameter) to enable country limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
CountryCode	(input parameter) the country code to lock the registration key to (see Enigma API EP_MiscCountryCode to get the code of a particular country);
UseRegisterAfter	(input parameter) to enable Register After limit for the registration key, set this value to 1, otherwise it should be 0. The Register After date is specified in the RegisterAfterYear, RegisterAfterMonth, RegisterAfterDay parameters (see Creating Keys);
RegisterAfterYear	(input parameter) year of Register After date;
RegisterAfterMonth	(input parameter) month of Register After date;
RegisterAfterDay	(input parameter) day of Register After date;
UseRegisterBefore	(input parameter) to enable Register Before limit for the registration key, set this value to 1, otherwise it should be 0. The Register Before date is specified in the RegisterBeforeYear, RegisterBeforeMonth, RegisterBeforeDay parameters (see Creating Keys);
RegisterBeforeYear	(input parameter) year of Register Before date;
RegisterBeforeMonth	(input parameter) month of Register Before date;
RegisterBeforeDay	(input parameter) day of Register Before date;
EncryptedConstant	(input parameter) integer constant, obtain it from the project file (see Registration Features - Common Registration, Information for Custom Keys Generator box, Encryption Constant value);
EncryptedSections	(input parameter) array of 16-byte length, which points to the encrypted sections, that will be decrypted by this key (see Creating Keys);
PublicKey	(input parameter) pointer to ANSI, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Public Key value);
PrivateKey	(input parameter) pointer to ANSI, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Private Key value).

Definition

[+ Show/Hide Delphi structure definition](#)

[+ Show/Hide C++ structure definition](#)

[+ Show/Hide C# \(.NET\) structure definition](#)

Function KG_GenerateRegistrationKeyA

The KG_GenerateRegistrationKeyA function is almost the same as the [KG_GenerateRegistrationKey](#) function and is used for ANSI-based registration key generation. Note: to enable the application to accept ANSI registration keys, the UNICODE Registration Scheme parameter from the [Registration Features - Common](#) panel should be disabled. The function has one parameter - PKeyGenParamsA, which is a pointer to the [TGenKeyParamsA](#) structure. This function requires the following necessary input parameters: PublicKey, PrivateKey, KeyMode, KeyBase and EncryptedConstant (other parameters are optional) that should be obtained from the project (see the [Registration Features - Common](#) panel, Information for Custom Generator box).

Return Values

EP_NO_ERROR=0	the function succeeds. In this case, Key parameter of TKeyGenParams structure should contain a pointer to the null terminated string - registration key.
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Keygen subfolder.

Function KG_GenerateRegistrationKeyFromProjectA

KG_GenerateRegistrationKeyFromProjectA returns the same result as the [KG_GenerateRegistrationKeyFromProject](#) function and is used for generating ANSI-based registration keys. The main difference of this function is that it reads secure information (PublicKey, PrivateKey, KeyMode, KeyBase and EncryptedConstant) from the project file and you do not have to set up these parameters in the input structure [TGenKeyParamA](#). KG_GenerateRegistrationKeyFromProjectA has two parameters, one of them is ProjectFile, which is a pointer to ANSI - null terminated string - the path to the Enigma Protector project file, and another parameter is a pointer to the [TGenKeyParamA](#) structure.

Warning: ProjectFile has to contain the absolute path to the project file.

Return Values

EP_NO_ERROR=0	the function succeeds. In this case, Key parameter of TKeyGenParams structure should contain a pointer to the null terminated string - registration key.
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Keygen subfolder.

TKeyGenParamsW type

TKeyGenParamsW structure is used by KG_GenerateRegistrationKeyW and KG_GenerateRegistrationKeyFromProjectW to generate registration keys.

Structure description

KeyMode	(input parameter) mode of registration key (RSA 512/768/1024/2048/3072/4096). It may have the following values (see Registration Features - Common Registration key safety/length value): RM_512 = 0; RM_768 = 1; RM_1024 = 2; RM_2048 = 3; RM_3072 = 4; RM_4096 = 5;
KeyBase	(input parameter) base of registration key (Base 2/8/16/32/64). It may have the following values (see Registration Features - Common Registration key output base value): RB_2 = 0; RB_8 = 1; RB_16 = 2; RB_32 = 3; RB_64 = 4;
KeyWithHyphens	(input parameter) add hyphens to the key, set to 1 if true or 0 if false (see Creating Keys);
Key	(input parameter) pointer to the memory where the registration key will be placed. The registration key will be represented as unicode, null terminated string. Please be sure that the buffer allocated for the registration key has sufficient size, we recommend allocating a buffer at least 4096 bytes long;
KeyLen	(input parameter) size of registration key buffer;
RegInfo	(input parameter) pointer to unicode, null terminated string, which is the registration name;
UseKeyExpiration	(input parameter) if the value is 0, then the key does not have expiration, if the value is 1, then the key is time-limited. The key will expire on the date specified in the ExpirationYear, ExpirationMonth, ExpirationDay parameters (see Creating Keys);
ExpirationYear	(input parameter) year of key expiration date;
ExpirationMonth	(input parameter) month of key expiration date;
ExpirationDay	(input parameter) day of key expiration date;
UseHardwareLocking	(input parameter) if the key is hardware-locked, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
HardwareIDType	(input parameter) pointer to ANSI, null terminated hardware id string;
UseExecutionsLimit	(input parameter) to enable executions limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
ExecutionsCount	(input parameter) number of executions to limit the registration key to;
UseDaysLimit	(input parameter) to enable days limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
DaysCount	(input parameter) number of days to limit the registration key to;
UseRunTimeLimit	(input parameter) to enable run-time limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
RunTimeMinutes	(input parameter) number of run-time minutes to limit the registration key to;

UseGlobalTimeLimit	(input parameter) to enable global time limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
GlobalTimeMinutes	(input parameter) number of global time minutes to limit the registration key to;
UseCountyLimit	(input parameter) to enable country limit, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
CountryCode	(input parameter) the country code to limit the registration key to (see Enigma API EP_MiscCountryCode to get the code of a particular country);
UseRegisterAfter	(input parameter) to enable Register After limit for registration key, set this value to 1, otherwise it should be 0. The Register After date is specified in the RegisterAfterYear, RegisterAfterMonth, RegisterAfterDay parameters (see Creating Keys);
RegisterAfterYear	(input parameter) year of Register After date;
RegisterAfterMonth	(input parameter) month of Register After date;
RegisterAfterDay	(input parameter) day of Register After date;
UseRegisterBefore	(input parameter) to enable Register Before limit for registration key, set this value to 1, otherwise it should be 0. The Register Before date is specified in the RegisterBeforeYear, RegisterBeforeMonth, RegisterBeforeDay parameters (see Creating Keys);
RegisterBeforeYear	(input parameter) year of Register Before date;
RegisterBeforeMonth	(input parameter) month of Register Before date;
RegisterBeforeDay	(input parameter) day of Register Before date;
EncryptedConstant	(input parameter) integer constant, obtain it from the project file (see Registration Features - Common Registration, Information for Custom Keys Generator box, Encryption Constant value);
EncryptedSections	(input parameter) array of 16-byte length, which points to the encrypted sections, that will be decrypted by this key (see Creating Keys);
PublicKey	(input parameter) pointer to unicode, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Public Key value);
PrivateKey	(input parameter) pointer to unicode, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Private Key value).

Definition

[+ Show/Hide Delphi structure definition](#)

[+ Show/Hide C++ structure definition](#)

[+ Show/Hide C# \(.NET\) structure definition](#)

Function KG_GenerateRegistrationKeyW

KG_GenerateRegistrationKeyW, as well as the [KG_GenerateRegistrationKey](#) function, is used for generating registration keys, but it uses unicode-based information. Note: to enable the application to accept unicode registration keys, the UNICODE Registration Scheme parameter from the [Registration Features - Common](#) panel should be enabled. The function has one parameter - PKeyGenParamsA, which is a pointer to the [TGenKeyParamsW](#) structure. This function requires the following necessary input parameters: PublicKey, PrivateKey, KeyMode, KeyBase and EncryptedConstant (other parameters are optional), that should be obtained from the the project (see the [Registration Features - Common](#) panel, Information for Custom Generator box).

Return Values

EP_NO_ERROR=0	the function succeeds. In this case, Key parameter of TKeyGenParams structure should contain a pointer to the null terminated string - registration key.
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

Examples

[+ Show/Hide Delphi function example](#)

[+ Show/Hide Visual C++ function example](#)

[+ Show/Hide C# \(.NET\) function example](#)

See function examples in the installation folder, Examples\KeygenUnicode subfolder.

Function KG_GenerateRegistrationKeyFromProjectW

KG_GenerateRegistrationKeyFromProjectW, as well as the [KG_GenerateRegistrationKeyW](#) function, is used for generating unicode-based registration keys. Note: to enable the application to accept unicode registration keys, the UNICODE Registration Scheme parameter from the [Registration Features - Common](#) panel should be enabled. The function reads secure information (PublicKey, PrivateKey, KeyMode, KeyBase and EncryptedConstant) from the project file and you do not have to set up these parameters in the input structure [TGenKeyParamW](#). KG_GenerateRegistrationKeyFromProjectW has two parameters, one of them is ProjectFile, which is a pointer to unicode - null terminated string - the path to the Enigma Protector project file, and another parameter is a pointer to the [TGenKeyParamW](#) structure.

Return Values

EP_NO_ERROR=0	the function succeeds. In this case, Key parameter of TKeyGenParams structure should contain a pointer to the null terminated string - registration key.
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

Examples

[+ Show/Hide Delphi function example](#)

[+ Show/Hide Visual C++ function example](#)

[+ Show/Hide C# \(.NET\) function example](#)

See function examples in the installation folder, Examples\KeygenUnicode subfolder.

TKeyVerifyParams type

TKeyVerifyParams structure is used by [KG_VerifyRegistrationInfo](#) and [KG_VerifyRegistrationInfoFromProject](#) to verify registration information.

Structure description

KeyMode	(input parameter) mode of registration key (RSA 512/768/1024/2048/3072/4096). It may have the following values (see Registration Features - Common Registration key safety/length value): RM_512 = 0; RM_768 = 1; RM_1024 = 2; RM_2048 = 3; RM_3072 = 4; RM_4096 = 5;
KeyBase	(input parameter) base of registration key (Base 2/8/16/32/64). It may have the following values (see Registration Features - Common Registration key output base value): RB_2 = 0; RB_8 = 1; RB_16 = 2; RB_32 = 3; RB_64 = 4;
Key	(input parameter) pointer to the memory where the registration key will be placed. The registration key will be represented as ANSI, null terminated string. Please be sure that the buffer allocated for the registration key has sufficient size, we recommend allocating a buffer at least 2048 bytes long;
KeyLen	(input parameter) size of registration key buffer;
RegInfo	(input parameter) pointer to the memory where the registration name info is placed ;
RegInfoLen	(input parameter) size of registration info;
CreationYear	(output parameter) the year when the key was created
CreationMonth	(output parameter) the month when the key was created
CreationDay	(output parameter) the day when the key was created
UseKeyExpiration	(output parameter) if the value is 0, then the key does not have expiration, if the value is 1, then the key is time-limited. The key will expire on the date specified in the ExpirationYear, ExpirationMonth, ExpirationDay parameters (see Creating Keys);
ExpirationYear	(output parameter) year of key expiration date;
ExpirationMonth	(output parameter) month of key expiration date;
ExpirationDay	(output parameter) day of key expiration date;
UseHardwareLocking	(input parameter) if the key is hardware-locked, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
HardwareIDType	(input parameter) pointer to ANSI, null terminated hardware id string;
UseExecutionsLimit	(output parameter) key is executions-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
ExecutionsCount	(output parameter) number of executions the registration key is limited to;
UseDaysLimit	(output parameter) key is days-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
DaysCount	(output parameter) number of days the registration key is limited to;
UseRunTimeLimit	(output parameter) key is run-time-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);

RunTimeMinutes	(output parameter) number of run-time minutes the registration key is limited to;
UseGlobalTimeLimit	(output parameter) key is global time-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
GlobalTimeMinutes	(output parameter) number of global time minutes the registration key is limited to;
UseCountyLimit	(output parameter) key is country-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
CountryCode	(output parameter) the country code the registration key is limited to (see Enigma API EP_MiscCountryCode to get the code of a particular country);
UseRegisterAfter	(output parameter) the registration key has Register After date; the parameter is 1 if true, and 0 if false. The Register After date is specified in the RegisterAfterYear, RegisterAfterMonth, RegisterAfterDay parameters (see Creating Keys);
RegisterAfterYear	(output parameter) year of Register After date;
RegisterAfterMonth	(output parameter) month of Register After date;
RegisterAfterDay	(output parameter) day of Register After date;
UseRegisterBefore	(output parameter) the registration key has Register Before date; the parameter is 1 if true, and 0 if false. The Register Before date is specified in the RegisterBeforeYear, RegisterBeforeMonth, RegisterBeforeDay parameters (see Creating Keys);
RegisterBeforeYear	(output parameter) year of Register Before date;
RegisterBeforeMonth	(output parameter) month of Register Before date;
RegisterBeforeDay	(output parameter) day of Register Before date;
EncryptedConstant	(input parameter) integer constant, obtain it from the project file (see Registration Features - Common Registration, Information for Custom Keys Generator box, Encryption Constant value);
EncryptedSections	(output parameter) array of 16-byte length, which shows the encrypted sections, that will be decrypted with this registration key (see Creating Keys);
PublicKey	(input parameter) pointer to ANSI, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Public Key value);

Definition

- [⊕ Show/Hide Delphi structure definition](#)
- [⊕ Show/Hide C++ structure definition](#)
- [⊕ Show/Hide C# \(.NET\) structure definition](#)
- [⊕ Show/Hide Visual Basic structure definition](#)

Function KG_VerifyRegistrationInfo

KG_VerifyRegistrationInfo is used for verifying the registration information and extracting registration key parameters from ANSI-based registration information. The function has one parameter - a pointer to the [TKeyVerifyParams](#) structure. Please note, the [TKeyVerifyParams](#) structure should contain secure parameters from the Enigma Protector project. These parameters are PublicKey, KeyMode, KeyBase and EncryptedConstant (see the [Registration Features - Common](#) panel, Information for Custom Generator box).

Return Values

EP_NO_ERROR=0	the function succeeds, TKeyVerifyParams structure contains registration key parameters;
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

[+ Show/Hide Visual Basic function definition](#)

Examples

[+ Show/Hide Delphi function example](#)

[+ Show/Hide Borland C++ Builder function example](#)

[+ Show/Hide Visual C++ function example](#)

[+ Show/Hide C# \(.NET\) function example](#)

[+ Show/Hide Visual Basic function example](#)

See function examples in the installation folder, Examples\Keygen subfolder.

Function KG_VerifyRegistrationInfoFromProject

KG_VerifyRegistrationInfoFromProject is used for verifying the registration information and extracting registration key parameters from ANSI-based registration information. The function has two parameters, one of them is a pointer to the [TKeyVerifyParams](#) structure, the other one is ANSI - null terminated string - the path to the Enigma Protector project file. The function reads secure information from the project file, so it does not require secure parameters PublicKey, KeyMode, KeyBase and EncryptedConstant in the [TKeyVerifyParams](#) structure.

Return Values

EP_NO_ERROR=0	the function succeeds, TKeyVerifyParams structure contains registration key parameters;
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

[+ Show/Hide Visual Basic function definition](#)

Examples

[+ Show/Hide Delphi function example](#)

[+ Show/Hide Borland C++ Builder function example](#)

[+ Show/Hide Visual C++ function example](#)

[+ Show/Hide C# \(.NET\) function example](#)

[+ Show/Hide Visual Basic function example](#)

See function examples in the installation folder, Examples\Keygen subfolder.

TKeyVerifyParamsA type

TKeyVerifyParamsA structure is used by [KG_VerifyRegistrationInfoA](#) and [KG_VerifyRegistrationInfoFromProjectA](#) to verify registration information.

Structure description

KeyMode	(input parameter) mode of registration key (RSA 512/768/1024/2048/3072/4096). It may have the following values (see Registration Features - Common Registration key safety/length value): RM_512 = 0; RM_768 = 1; RM_1024 = 2; RM_2048 = 3; RM_3072 = 4; RM_4096 = 5;
KeyBase	(input parameter) base of registration key (Base 2/8/16/32/64). It may have the following values (see Registration Features - Common Registration key output base value): RB_2 = 0; RB_8 = 1; RB_16 = 2; RB_32 = 3; RB_64 = 4;
Key	(input parameter) pointer to the ANSI, null terminated string, which is the registration key;
KeyLen	(input parameter) size of registration key buffer;
RegInfo	(input parameter) pointer to ANSI, null terminated string, which is the registration name;
CreationYear	(output parameter) the year when the key was created
CreationMonth	(output parameter) the month when the key was created
CreationDay	(output parameter) the day when the key was created
UseKeyExpiration	(output parameter) if the value is 0, then the key does not have expiration, if the value is 1, then the key is time-limited. The key will expire on the date specified in the ExpirationYear, ExpirationMonth, ExpirationDay parameters (see Creating Keys);
ExpirationYear	(output parameter) year of key expiration date;
ExpirationMonth	(output parameter) month of key expiration date;
ExpirationDay	(output parameter) day of key expiration date;
UseHardwareLocking	(input parameter) if the key is hardware-locked, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
HardwareIDType	(input parameter) pointer to ANSI, null terminated hardware id string;
UseExecutionsLimit	(output parameter) key is executions-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
ExecutionsCount	(output parameter) number of executions the registration key is limited to;
UseDaysLimit	(output parameter) key is days-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
DaysCount	(output parameter) number of days the registration key is limited to;
UseRunTimeLimit	(output parameter) key is run-time-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
RunTimeMinutes	(output parameter) number of run-time minutes the registration key is limited to;
UseGlobalTimeLimit	(output parameter) key is global time-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
GlobalTimeMinutes	(output parameter) number of global time minutes the registration key is limited to;

UseCountyLimit	(output parameter) key is country-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
CountryCode	(output parameter) the country code the registration key is locked to (see Enigma API EP_MiscCountryCode to get the code of a particular country);
UseRegisterAfter	(output parameter) the registration key has Register After date; the parameter is 1 if true yes and 0 if false. The Register After date is specified in the RegisterAfterYear, RegisterAfterMonth, RegisterAfterDay parameters (see Creating Keys);
RegisterAfterYear	(output parameter) year of Register After date;
RegisterAfterMonth	(output parameter) month of Register After date;
RegisterAfterDay	(output parameter) day of Register After date;
UseRegisterBefore	(output parameter) the registration key has Register Before date; the parameter is 1 if true, and 0 if false. The Register Before date is specified in the RegisterBeforeYear, RegisterBeforeMonth, RegisterBeforeDay parameters (see Creating Keys);
RegisterBeforeYear	(output parameter) year of Register Before date;
RegisterBeforeMonth	(output parameter) month of Register Before date;
RegisterBeforeDay	(output parameter) day of Register Before date;
EncryptedConstant	(input parameter) integer constant, obtain it from the project file (see Registration Features - Common Registration, Information for Custom Keys Generator box, Encryption Constant value);
EncryptedSections	(output parameter) array of 16-byte length, which the encrypted sections, that will be decrypted with this registration key (see Creating Keys);
PublicKey	(input parameter) pointer to ANSI, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Public Key value);

Definition

[+ Show/Hide Delphi structure definition](#)

[+ Show/Hide C++ structure definition](#)

[+ Show/Hide C# \(.NET\) structure definition](#)

Function KG_VerifyRegistrationInfoA

KG_VerifyRegistrationInfoA, as well as the [KG_VerifyRegistrationInfo](#) function, is used for verifying the registration information and extracting registration key parameters from ANSI-based registration information. The function has one parameter - a pointer to the [TKeyVerifyParamsA](#) structure. Please note, the [TKeyVerifyParamsA](#) structure should contain secure parameters from the Enigma Protector project. These parameters are PublicKey, KeyMode, KeyBase and EncryptedConstant (see the [Registration Features - Common](#) panel, Information for Custom Generator box).

Return Values

EP_NO_ERROR=0	the function succeeds, TKeyVerifyParamsA structure contains registration key parameters;
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Keygen subfolder.

Function KG_VerifyRegistrationInfoFromProjectA

KG_VerifyRegistrationInfoFromProjectA is used for verifying the registration information and extracting registration key parameters from ANSI-based registration information. The function has two parameters, one of them is a pointer to the [TKeyVerifyParamsA](#) structure, the other one is ANSI - null terminated string - the path to the Enigma Protector project file. The function reads secure information from the project file, so it does not require secure parameters PublicKey, KeyMode, KeyBase and EncryptedConstant in the [TKeyVerifyParamsA](#) structure.

Return Values

EP_NO_ERROR=0	the function succeeds, TKeyVerifyParamsA structure contains registration key parameters;
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

See function examples in the installation folder, Examples\Keygen subfolder.

TKeyVerifyParamsW type

TKeyVerifyParamsW structure is used by [KG_VerifyRegistrationInfoW](#) and [KG_VerifyRegistrationInfoFromProjectW](#) to verify registration information.

Structure description

KeyMode	(input parameter) mode of registration key (RSA 512/768/1024/2048/3072/4096). It may have the following values (see Registration Features - Common Registration key safety/length value): RM_512 = 0; RM_768 = 1; RM_1024 = 2; RM_2048 = 3; RM_3072 = 4; RM_4096 = 5;
KeyBase	(input parameter) base of registration key (Base 2/8/16/32/64). It may have the following values (see Registration Features - Common Registration key output base value): RB_2 = 0; RB_8 = 1; RB_16 = 2; RB_32 = 3; RB_64 = 4;
Key	(input parameter) pointer to the unicode, null terminated string, which is the registration key;
KeyLen	(input parameter) size of registration key buffer;
RegInfo	(input parameter) pointer to unicode, null terminated string, which is the registration name;
CreationYear	(output parameter) the year when the key was created
CreationMonth	(output parameter) the month when the key was created
CreationDay	(output parameter) the day when the key was created
UseKeyExpiration	(output parameter) if the value is 0, then the key does not have expiration, if the value is 1, then the key is time-limited. The key will expire on the date specified in the ExpirationYear, ExpirationMonth, ExpirationDay parameters (see Creating Keys);
ExpirationYear	(output parameter) year of key expiration date;
ExpirationMonth	(output parameter) month of key expiration date;
ExpirationDay	(output parameter) day of key expiration date;
UseHardwareLocking	(input parameter) if the key is hardware-locked, set this parameter to 1, otherwise it should be 0 (see Creating Keys);
HardwareIDType	(input parameter) pointer to unicode, null terminated hardware id string;
UseExecutionsLimit	(output parameter) key is executions-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
ExecutionsCount	(output parameter) number of executions the registration key is limited to;
UseDaysLimit	(output parameter) key is days-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
DaysCount	(output parameter) number of days the registration key is limited to;
UseRunTimeLimit	(output parameter) key is run-time-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
RunTimeMinutes	(output parameter) number of run-time minutes the registration key is limited to;
UseGlobalTimeLimit	(output parameter) key is global time-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
GlobalTimeMinutes	(output parameter) number of global time minutes the registration key is limited to;

UseCountyLimit	(output parameter) key is country-limited; the parameter is 1 if the key is limited, and 0 if it is not (see Creating Keys);
CountryCode	(output parameter) the country code the registration key is locked to (see Enigma API EP_MiscCountryCode to get the code of a particular country);
UseRegisterAfter	(output parameter) the registration key has Register After date; the parameter is 1 if true, and 0 if false. The Register After date is specified in the RegisterAfterYear, RegisterAfterMonth, RegisterAfterDay parameters (see Creating Keys);
RegisterAfterYear	(output parameter) year of Register After date;
RegisterAfterMonth	(output parameter) month of Register After date;
RegisterAfterDay	(output parameter) day of Register After date;
UseRegisterBefore	(output parameter) the registration key has Register Before date; the parameter is 1 if true, and 0 if false. The Register Before date is specified in the RegisterBeforeYear, RegisterBeforeMonth, RegisterBeforeDay parameters (see Creating Keys);
RegisterBeforeYear	(output parameter) year of Register Before date;
RegisterBeforeMonth	(output parameter) month of Register Before date;
RegisterBeforeDay	(output parameter) day of Register Before date;
EncryptedConstant	(input parameter) integer constant, obtain it from the project file (see Registration Features - Common Registration, Information for Custom Keys Generator box, Encryption Constant value);
EncryptedSections	(output parameter) array of 16-byte length, which shows the encrypted, that will be decrypted with this registration key (see Creating Keys);
PublicKey	(input parameter) pointer to unicode, null terminated string, which is a unique constant, should be obtained from the project (see Registration Features - Common Registration, Information for Custom Keys Generator box, Public Key value);

Definition

[+ Show/Hide Delphi structure definition](#)

[+ Show/Hide C++ structure definition](#)

[+ Show/Hide C# \(.NET\) structure definition](#)

Function KG_VerifyRegistrationInfoW

KG_VerifyRegistrationInfoW is used for verifying the registration information and extracting registration key parameters from unicode-based registration information. The function has one parameter - a pointer to the [TKeyVerifyParamsW](#) structure. Please note, the [TKeyVerifyParamsW](#) structure should contain secure parameters from the Enigma Protector project. These parameters are PublicKey, KeyMode, KeyBase and EncryptedConstant (see the [Registration Features - Common](#) panel, Information for Custom Generator box).

Return Values

EP_NO_ERROR=0	the function succeeds, TKeyVerifyParamsW structure contains registration key parameters;
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

Examples

[+ Show/Hide Delphi function example](#)

[+ Show/Hide Visual C++ function example](#)

[+ Show/Hide C# \(.NET\) function example](#)

See function examples in the installation folder, Examples\KeygeUnicode subfolder.

Function KG_VerifyRegistrationInfoFromProjectW

KG_VerifyRegistrationInfoFromProjectW is used for verifying the registration information and extracting registration key parameters from unicode-based registration information. The function has two parameters, one of them is a pointer to the [TKeyVerifyParamsW](#) structure, the other one is unicode - null terminated string - the path to the Enigma Protector project file. The function reads secure information from the project file, so it does not require secure parameters PublicKey, KeyMode, KeyBase and EncryptedConstant in the [TKeyVerifyParamsW](#) structure.

Return Values

EP_NO_ERROR=0	the function succeeds, TKeyVerifyParamsW structure contains registration key parameters;
EP_ERROR_UNKNOWN=1	unknown error
EP_ERROR_KEYBUFFEREMPTY=2	memory buffer for registration key is not allocated
EP_ERROR_KEYBUFFERISLESS=3	size of allocated memory for registration key is less than required
EP_ERROR_REGINFOEMPTY=4	registration information is not specified
EP_ERROR_REGINFOTOOLARGE=5	registration information is empty (has a null size)
EP_ERROR_PRIVATEKEYISNOTSET=6	private key is not set
EP_ERROR_PUBLICKEYISNOTSET=7	public key is not set
EP_ERROR_PRIVATEKEYISINVALID=8	private key is invalid
EP_ERROR_PUBLICKEYISINVALID=9	public key is invalid
EP_ERROR_KEYMODEISINVALID=10	key mode is invalid
EP_ERROR_KEYBASEISINVALID=11	key base is invalid
EP_ERROR_CURRENTDATEISINVALID=12	current date is invalid
EP_ERROR_EXPIRATIONDATEISINVALID=13	expiration date is invalid
EP_ERROR_KEYISINVALID=14	key is invalid
EP_ERROR_HARDWAREID=15	hardware id is invalid
EP_ERROR_HARDWAREBUFFEREMPTY=16	hardware id buffer is empty
EP_ERROR_HARDWAREIDINVALIDFORKEY=17	hardware id is invalid for the key
EP_ERROR_PROJECTFILENOTFOUND=18	project file is not found
EP_ERROR_INVALIDPROJECTFILE=19	project file is invalid
EP_ERROR_EXECUTIONSNUMBERINVALID=20	executions number is invalid
EP_ERROR_DAYSNUMBERINVALID=21	days number is invalid
EP_ERROR_COUNTRYCODEINVALID=22	country code is invalid
EP_ERROR_RUNTIMEINVALID=23	run-time value is invalid
EP_ERROR_GLOBALTIMEINVALID=24	global time is invalid
EP_ERROR_INSTALLBEFOREINVALID=25	register before date is invalid
EP_ERROR_INSTALLAFTERINVALID=26	register after date is invalid

Definition

[+ Show/Hide Delphi function definition](#)

[+ Show/Hide C++ function definition](#)

[+ Show/Hide C# \(.NET\) function definition](#)

Examples

[+ Show/Hide Delphi function example](#)

[+ Show/Hide Visual C++ function example](#)

[+ Show/Hide C# \(.NET\) function example](#)

See function examples in the installation folder, Examples\KeygenUnicode subfolder.

Markers

Markers are special inclusions in the source code of the module that allow adding a wide range of possibilities to increase safety of the protected module. The functionality of an unprotected module will not be affected by markers. To activate markers you need to protect the module before using it.

Please Note:

- | Markers cannot be used in .NET applications;
- | Visual Basic 6 applications should be compiled in native mode (not pcode).

The Enigma Protector supports the following kinds of markers:

- | [VM markers](#) - allow selecting code that will be emulated after the protection and executed under its own Virtual Processor. This marker is the best choice to protect weak points of the protection/registration routines.
- | [Reg_Crypt markers](#) - allow selecting parts of the code that should only be executed in a registered version
- | [UnReg_Crypt markers](#) - allow selecting parts of the code that should only be executed in an unregistered version
- | [Decrypt_On_Execute markers](#) - allow selecting parts of the code that should only be deciphered when the code needs to be executed
- | [Unprotected markers](#) - allow selecting the code that should not be executed in a protected version of a file
- | [Check_Protection markers](#) - this markers check the integrity of the protection code, and if it is corrupted, the code inside markers will not be executed
- | [Run_Once markers](#) - the selected code in this marker will be run only once while execution, after the first execution the marker will be deleted from memory and will never be executed again

Rules of markers usage

A Marker is a set of bytes placed into the source code and helping Enigma Protector find the code inside markers for processing. A marker consists of two parts: begin marker and end marker. You should remember the following rules of markers usage:

- 1 The end marker can't be placed before the begin marker, otherwise it will cause an error and all markers will be ignored.

```
{WRONG}
{$I include\reg_crypt_end1.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
                #10#13'and section #1 unlocked by registration key' +
                #10#13, 'Application', 0);
{$I include\reg_crypt_begin1.inc}

{RIGHT}
{$I include\reg_crypt_begin1.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
                #10#13'and section #1 unlocked by registration key' +
                #10#13, 'Application', 0);
{$I include\reg_crypt_end1.inc}
```

- 1 The markers should be placed in pairs, i.e. there should be both begin and end marker. In case you have lost one marker from the pair, all markers will be ignored.

```
{WRONG}
{$I include\reg_crypt_begin1.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
                #10#13'and section #1 unlocked by registration key' +
                #10#13, 'Application', 0);
{$I include\reg_crypt_begin2.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
                #10#13'and section #1 unlocked by registration key' +
                #10#13, 'Application', 0);
{$I include\reg_crypt_end2.inc}

{RIGHT}
{$I include\reg_crypt_begin1.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
                #10#13'and section #1 unlocked by registration key' +
                #10#13, 'Application', 0);
{$I include\reg_crypt_end1.inc}
{$I include\reg_crypt_begin2.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
                #10#13'and section #1 unlocked by registration key' +
                #10#13, 'Application', 0);
{$I include\reg_crypt_end2.inc}
```

- 1 Markers should not contain any other kind of markers.

```
{WRONG}
{$I include\reg_crypt_begin1.inc}
    {$I include\reg_crypt_begin2.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
                #10#13'and section #1 unlocked by registration key' +
                #10#13, 'Application', 0);
    {$I include\reg_crypt_end2.inc}
{$I include\reg_crypt_end1.inc}

{RIGHT}
{$I include\reg_crypt_begin1.inc}
    MessageBox(0, #10#13'This message appears only if application is registered' +
```

```

        #10#13'and section #1 unlocked by registration key' +
        #10#13, 'Application', 0);
    {$I include\reg_crypt_end1.inc}

```

- Markers should separate completed parts of the code:

```

{WRONG}
{$I include\reg_crypt_begin1.inc}
if EP_RegLoadKey(pcUserInfo, pcKey) then
begin
    eName.Text := string(pcUserInfo);
    {$I include\reg_crypt_end1.inc}
    eKey.Text := string(pcKey);
    ShowMessage('Thanks for registration');
end;

```

```

{RIGHT}
{$I include\reg_crypt_begin1.inc}
if EP_RegLoadKey(pcUserInfo, pcKey) then
begin
    eName.Text := string(pcUserInfo);
    eKey.Text := string(pcKey);
    ShowMessage('Thanks for registration');
end;
{$I include\reg_crypt_end1.inc}

```

- statements like If-Then-Else, For-To-Do, While-Do, Repeat-Until, Try-Except, Try-Finally etc. should be fully separated:

```

{WRONG}
if EP_RegLoadKey(pcUserInfo, pcKey) then
begin
    {$I include\reg_crypt_begin1.inc}
    eName.Text := string(pcUserInfo);
    eKey.Text := string(pcKey);
    ShowMessage('Thanks for registration');
    {$I include\reg_crypt_end1.inc}
end;

```

```

{RIGHT}
{$I include\reg_crypt_begin1.inc}
if EP_RegLoadKey(pcUserInfo, pcKey) then
begin
    eName.Text := string(pcUserInfo);
    eKey.Text := string(pcKey);
    ShowMessage('Thanks for registration');
end;
{$I include\reg_crypt_end1.inc}

```

- for C and C++ users: the return; keyword can't be placed inside the marker:

```

{WRONG}
void CTestDlg::CheckRegistered(BOOL bReg)
{
    CWnd* wnd;
    // Enable/disable unregister button
    wnd = GetDlgItem(IDC_BUTTON_UNREGISTER);
    wnd->EnableWindow(bReg);

    // Enable/disable unregister static
    wnd = GetDlgItem(IDC_BUTTON_REGISTER);
    wnd->EnableWindow(!bReg);
    // Enable/disable user info edit
    wnd = GetDlgItem(IDC_EDIT_USERINFO);
}

```

```

wnd->EnableWindow(!bReg);
// Enable/disable key edit
wnd = GetDlgItem(IDC_EDIT_KEY);
wnd->EnableWindow(!bReg);

char* sName = NULL;
char* sKey = NULL;
{$I include\check_protection_begin.inc}
if (bReg)
{
    if (EP_RegLoadKey(&sName, &sKey))
    {
        SetDlgItemText(IDC_EDIT_USERINFO, sName);
        SetDlgItemText(IDC_EDIT_KEY, sKey);
        return;
    }
}
{$I include\check_protection_end.inc}
}

```

```

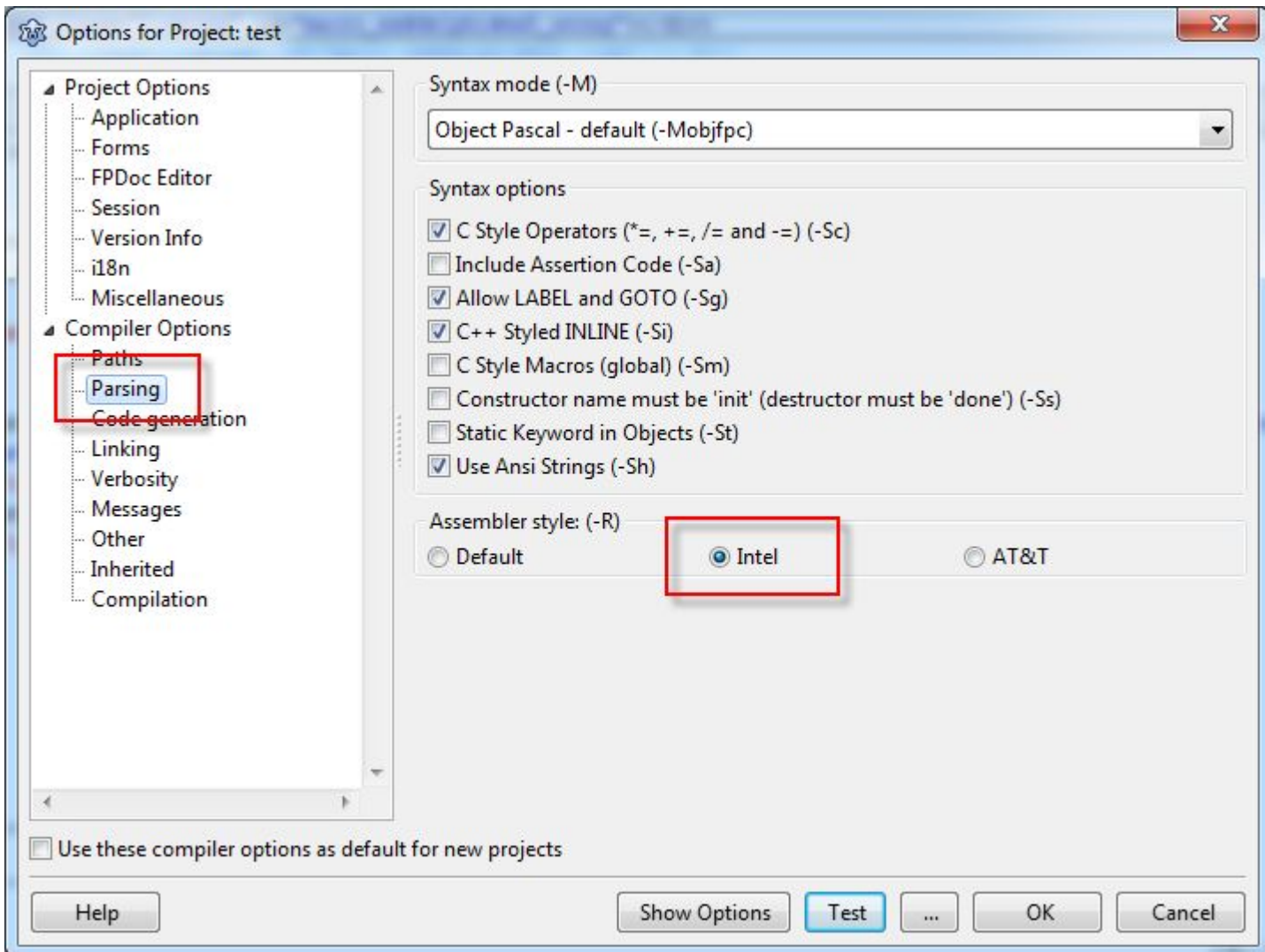
{RIGHT}
void CTestDlg::CheckRegistered(BOOL bReg)
{
    CWnd* wnd;
    // Enable/disable unregister button
    wnd = GetDlgItem(IDC_BUTTON_UNREGISTER);
    wnd->EnableWindow(bReg);

    // Enable/disable unregister static
    wnd = GetDlgItem(IDC_BUTTON_REGISTER);
    wnd->EnableWindow(!bReg);
    // Enable/disable user info edit
    wnd = GetDlgItem(IDC_EDIT_USERINFO);
    wnd->EnableWindow(!bReg);
    // Enable/disable key edit
    wnd = GetDlgItem(IDC_EDIT_KEY);
    wnd->EnableWindow(!bReg);

    char* sName = NULL;
    char* sKey = NULL;
    {$I include\check_protection_begin.inc}
    if (bReg)
    {
        if (EP_RegLoadKey(&sName, &sKey))
        {
            SetDlgItemText(IDC_EDIT_USERINFO, sName);
            SetDlgItemText(IDC_EDIT_KEY, sKey);
        }
    }
    {$I include\check_protection_end.inc}
    return;
}

```

- Free Pascal users should enable Intel Assembler Style in Compiler Options.



Markers VM

VM marker allows selecting parts of the code that will be analysed, virtualized and emulated. Virtualization means that the selected code will be translated into the PCODE (special programming language, which can only be read by Enigma) and executed under a Virtual Processor, and the application will require it. This cutting-edge technique allows protecting the weak points of your application that could be easily patched or analysed by crackers. The virtualized code is almost impossible to analyse for crackers, which makes the protection of your application much stronger. We recommended using as many of these markers as possible. Just note that the virtualized code will be executed more slowly than the original one, it is not recommended to virtualize extensively used parts of the code.

Content

- [⊕ Show/Hide Delphi marker content](#)
- [⊕ Show/Hide C++ marker content](#)
- [⊕ Show/Hide C++ x64 marker content](#)
- [⊕ Show/Hide Visual Basic marker content](#)

Examples

- [⊕ Show/Hide Delphi marker example](#)

See markers examples in the installation folder, Examples\MarkerVM subfolder.

Markers Reg_Crypt

Reg_Crypt markers allow selecting parts of the code that should be executed only in case the module is registered, and the registration key allows deciphering the crypted section. Enigma Protector supports 16 types of Reg_Crypt markers. Each marker can be used to select any number of the code parts. Note that the application has to be restarted after registration for the Reg_Crypt markers to work correctly. Restarting is not needed if you use the standard [Registration Dialog](#).

The principle of work: after the module is protected, the code inside the begin/end markers will be encrypted and the execution of the code will be impossible. If you try to run the protected module, the code inside the markers will not be executed, but other functionality will stay the same. If a valid registration key is stored in the system, the loader checks what sections can be decrypted with this key, and if succeeds, the necessary encrypted section will be decrypted. I.e. if the registration key was generated with Section 1 unlocked, the reg_crypt_begin1/reg_crypt_end1 marker will be decrypted. To register the module, use internal Enigma API functions or perform registration by means of key files, or use the standard [Registration Dialog](#). For detailed information on the registration key generator see [Creating Keys](#).

Content

- [+ Show/Hide Delphi marker content](#)
- [+ Show/Hide C++ marker content](#)
- [+ Show/Hide C++ x64 marker content](#)
- [+ Show/Hide Visual Basic marker content](#)

Examples

- [+ Show/Hide Delphi marker example](#)
- [+ Show/Hide C++ marker example](#)
- [+ Show/Hide Visual Basic marker example](#)

See markers examples in the installation folder, Examples\MarkersRegCrypt subfolder.

Markers UnReg_Crypt

The UnReg_Crypt markers allow selecting parts of the code that should be executed only in case the module is not registered, or the module is registered but the registration key does not support deciphering of the current encrypted section. The Enigma Protector supports 16 types of UnReg_Crypt markers. Each marker can be used to select any number of the code parts. The principle of work: after the module is protected and started, The Enigma Protector loader will check the presence of a valid registration key for the module. If a registration key is not found, all unreg_crypt sections will be deciphered and prepared for execution. There is another case in which these sections will be deciphered - in case a valid registration key is found, but it does not allow deciphering the current unreg_crypt section. Generally, the unreg_crypt markers are the opposite of the [reg_crypt](#) markers.

Content

- [+ Show/Hide Delphi marker content](#)
- [+ Show/Hide C++ marker content](#)
- [+ Show/Hide C++ x64 marker content](#)
- [+ Show/Hide Visual Basic marker content](#)

Examples

- [+ Show/Hide Delphi marker example](#)
- [+ Show/Hide C++ marker example](#)
- [+ Show/Hide Visual Basic marker example](#)

See markers examples in the installation folder, Examples\UnRegCryptMarkers subfolder.

Markers Decrypt_On_Execute

Decrypt_On_Execute markers allow selecting parts of the code that should be deciphered only when the code needs to be executed. These markers do not affect the execution process and the functionality of the protected module. The code between these markers enciphers after protection and is only stored in the file. When the execution of the module has come to the begin marker, the code begins to decipher and execute. These markers can increase safety of the protected module and serve as an anti-dump technique. Most crackers dump the program from memory at the first execution command, and if you use the Decrypt_On_Execute markers, the cracker will not get the original code from memory and will not be able to restore the sources. You can use any number of Decrypt_On_Execute markers in your module.

Content

- [⊕ Show/Hide Delphi marker content](#)
- [⊕ Show/Hide C++ marker content](#)
- [⊕ Show/Hide C++ x64 marker content](#)
- [⊕ Show/Hide Visual Basic marker content](#)

Examples

- [⊕ Show/Hide Delphi marker example](#)
- [⊕ Show/Hide C++ marker example](#)
- [⊕ Show/Hide Visual Basic marker example](#)

See markers examples in the installation folder, Examples\MarkersDecryptOnExecute subfolder.

Markers Check_Protection

Check_Protection markers can be used to check the integrity of the protection code. If the protection code is corrupted, the code inside the marker will not be executed, and if the protection is OK, the code inside the marker will be executed. This is a very good and useful way to hide the "weak" points from crackers, for example, the key check routine, etc. can be placed inside these markers.

Content

- [⊕ Show/Hide Delphi marker content](#)
- [⊕ Show/Hide C++ marker content](#)
- [⊕ Show/Hide C++ x64 marker content](#)
- [⊕ Show/Hide Visual Basic marker content](#)

Examples

- [⊕ Show/Hide Delphi marker example](#)
- [⊕ Show/Hide Visual Basic marker example](#)

See markers examples in the installation folder, Examples\CheckProtection subfolder.

Markers Run_Once

The Run_Once marker allows selecting part of the code that will be executed only the first time the file is executed. After the first execution the contents of this marker will be deleted from memory and will never be restored. If you try to run this code for a second time, it will be skipped.

Content

- [⊕ Show/Hide Delphi marker content](#)
- [⊕ Show/Hide C++ marker content](#)
- [⊕ Show/Hide C++ x64 marker content](#)
- [⊕ Show/Hide Visual Basic marker content](#)

Examples

- [⊕ Show/Hide Delphi marker example](#)

See markers examples in the installation folder, Examples\MarkerRunOnce subfolder.

Markers Unprotected

The Unprotected markers allow selecting parts of the code that should be executed only in a non-protected version of the file. During the protection process, the code inside these markers is deleted and these parts will not be present in memory when the protected file starts. In short, the code inside these markers will not be executed after protection.

Content

- [⊕ Show/Hide Delphi marker content](#)
- [⊕ Show/Hide C++ marker content](#)
- [⊕ Show/Hide C++ x64 marker content](#)
- [⊕ Show/Hide Visual Basic marker content](#)

Examples

- [⊕ Show/Hide Delphi marker example](#)
- [⊕ Show/Hide Visual Basic marker example](#)

See markers examples in the installation folder, Examples\MarkersUnprotect subfolder.

Tutorials

The tutorials will help beginner users apply the necessary type of protection in a couple of minutes. Select the protection type you would like to have:

- | [Simple envelope protection](#)
- | [Trial protection part 1](#)
- | [Trial protection part 2](#)

Simple envelop protection

The simple envelop protection implies creation of a simple protection scheme based on the protection of the main executable of the module, and does not require to make any changes to the module sources. The protection only uses the internal functions of The Enigma Protector, that prevent file disassembling, reverse engineering, patching (changing file and memory contents), memory dumping. The internal parameter for the project is only the executable file of your module.

- 1 Create a unique project for file protection. See. [Creating a protection project](#). This project can be used for creating more complex protection later.
- 1 Define the following settings: Input parameters, Protection parameters. Note: options related to the registration keys features and trial control features should not be used in the current project, so do not change these, leave them by default.
- 1 Module protection. Hit the "Protect" button and check the functionality of the protected module.

Delphi Examples

The sources below will guide you how to create a simple application displaying the "Hello World" message after execution. Note: the functionality of the module should not be changed after protection.

The full sources of the current project are located in the "Tutorials\Simple envelope protection\Delphi\" folder.

```
program test;
{$APPTYPE CONSOLE}
uses
  Windows, SysUtils;
begin
  MessageBox(0, 'This is The Enigma Protector test application.'#10#13 +
    'If after protection you will see this message then the application works correctly!',
    'Test Application', 0);
end.
```

Trial protection part 1

(without source changing)

Trial protection implies the use of specific trial options of The Enigma Protector to add time limitations to the protected module and use Enigma generated registration keys. The protection uses internal functions, so you do not need to make changes to the sources of your module. The only input parameter for the function is the executable file of your module.

1. Create a unique project for file protection. See. [Creation of protection project](#). This project can be later used for creating more complex protection.
2. Define the following settings: Input parameters, Protection parameters.
3. Choose trial period parameters. See. [Trial Limitations](#).
 - a. The Enigma Protector supports the following types of trial limitations: by count of module executions, by count of usage days (the trial period counter starts from the first execution of module on the user PC), by expiration date. Turn on the necessary additional parameters of the trial period.
 - b. Set the reaction on trial expiration. Trial expiration means the impossibility of future usage of the module on the user PC, so you can turn on the "Terminate after execution" option on the "Trial Control" panel (See. [Program Overview - Trial Control](#)) and enter the time interval after which the module will be terminated. If this value is 0, the module will be terminated immediately after it is started.
 - c. Trial period counter uses the system clock to calculate trial parameters. But a cracker can set the system clock backward to reset trial limitation and restore the functionality of the module. To prevent it, you can turn on the clock control function.
 - d. During the development of new software releases, you need to reset trial parameters of older protected versions of module on the user PC. To do this, you can enable a special feature which will check the version of trial information on the user's PC, and if it is older than the current version, the trial limitations will be reset. To enable the feature, check the "Reset trial" box on the "Trial control" panel. Note: this feature uses version information which you have entered into the project input parameters. See [Choosing input parameters](#)).
4. Set the registration keys features (See. [Registration keys features](#)). To delete trial limitations, the user needs to register your module. Module registration means saving the registration information on the user's PC. Choose the type and path for storing registration information. The Enigma Protector supports two types of registration information storage - registration information stored in an external file or in the Windows registry (See [Program Overview - Registration Features](#)).
5. Module protection. Click the "Protect" button and check the functionality of the protected module.

Overview:

After the first module execution on the user's PC, Enigma loader starts incrementing the trial counter. If the trial period has expired, the loader generates the reaction defined in 3.2. In case of trial expiration you can restore the functionality of the protected module by means of a file with registration information. You need to create a registration key by means of internal registration key generator (See. [Creating Keys](#)). Subject to the selected type of registration information storage (external file or Windows registry), you need to create the following file(s) with registration information:

- 1. registration information stored in the Windows registry.

Create any file with the .reg extension and place the following strings there:

```
[HKEY_ROOT\HKEY_RELATIVE]
"Name"="User Information"
"Key"="XUCBSNJ9T642DCQP3FW8C6NJL75X7"
```

here: HKEY_ROOT - root path of Windows registry, it can be:

HKEY_LOCAL_MACHINE and HKEY_CURRENT_USER, - this field has been filled out in the [Registration Features - Registration data storing](#) panel, "Base place in registry" field;

HKEY_RELATIVE - related path in the Windows registry, has been filled out in the "Relative path in registry"

field

User Information - information about the user the registration key has been generated for;

XUCBSNJ9T642DCQP3FW8C6N JL75X7 - the registration key.

Send the .reg file to the user. After the user double-clicks this file, a message will be displayed offering to add information into the Windows registry, the user must click "Yes". In case everything has been done successfully, the module will be registered.

registration information stored in the external file.

Create a file with the name which you have typed in the [Registration Features - Registration data storing](#), "Relative path in the file system" field and place the following information there:

```
[Registration information]
Name=UserInfo
Key=XUCBSNJ9T642DCQP3FW8C6N JL75X7
```

here UserInfo - information about the user the registration key has been generated for;

XUCBSNJ9T642DCQP3FW8C6N JL75X7 - the registration key.

Send this file to the user and instruct them to copy the registration file into the folder specified in the "Base folder in file system" field. In case everything has been done successfully, the module will be registered.

Delphi Examples

The sources below will guide you on how to create a simple application displaying a "Hello World" message after execution. Note: the functionality of the module should not be changed after protection. The full sources of the current project are located in the "Tutorials\Trial protection part 1\Delphi\" folder.

```
program test;
{$APPTYPE CONSOLE}
uses
  Windows,
  SysUtils;
begin
  MessageBox(0, 'This is The Enigma Protector test application.'#10#13 +
    'If after protection you will see this message then the application works correctly!',
    'Test Application', 0);
end.
```

Trial protection part 2 (Trial Enigma API's)

The protection is based on the limitation of the time period the module will work on the user's PC. The protection uses special Enigma API functions to get parameters of the trial period. After protection, you need to make the necessary changes to the sources of your module and provide the results of API call handling.

Changing the module sources.

The following internal Enigma API functions can be used to control trial parameters:

- | [EP_TrialExecutions](#)
- | [EP_TrialDays](#)
- | [EP_TrialExpirationDate](#)

The examples showing how to work with Enigma API trial functions for different development languages are located in the `Tutorials\Trial protection part 2\` folder.

The main advantage of this protection option is manual handling of trial parameters during the work of the module. You can change custom events on trial parameters alterations and trial expiration.

The following actions for project creating, setting of protection parameters, creation and distribution of registration keys are similar to the [Trial protection part 1](#).

Delphi Examples

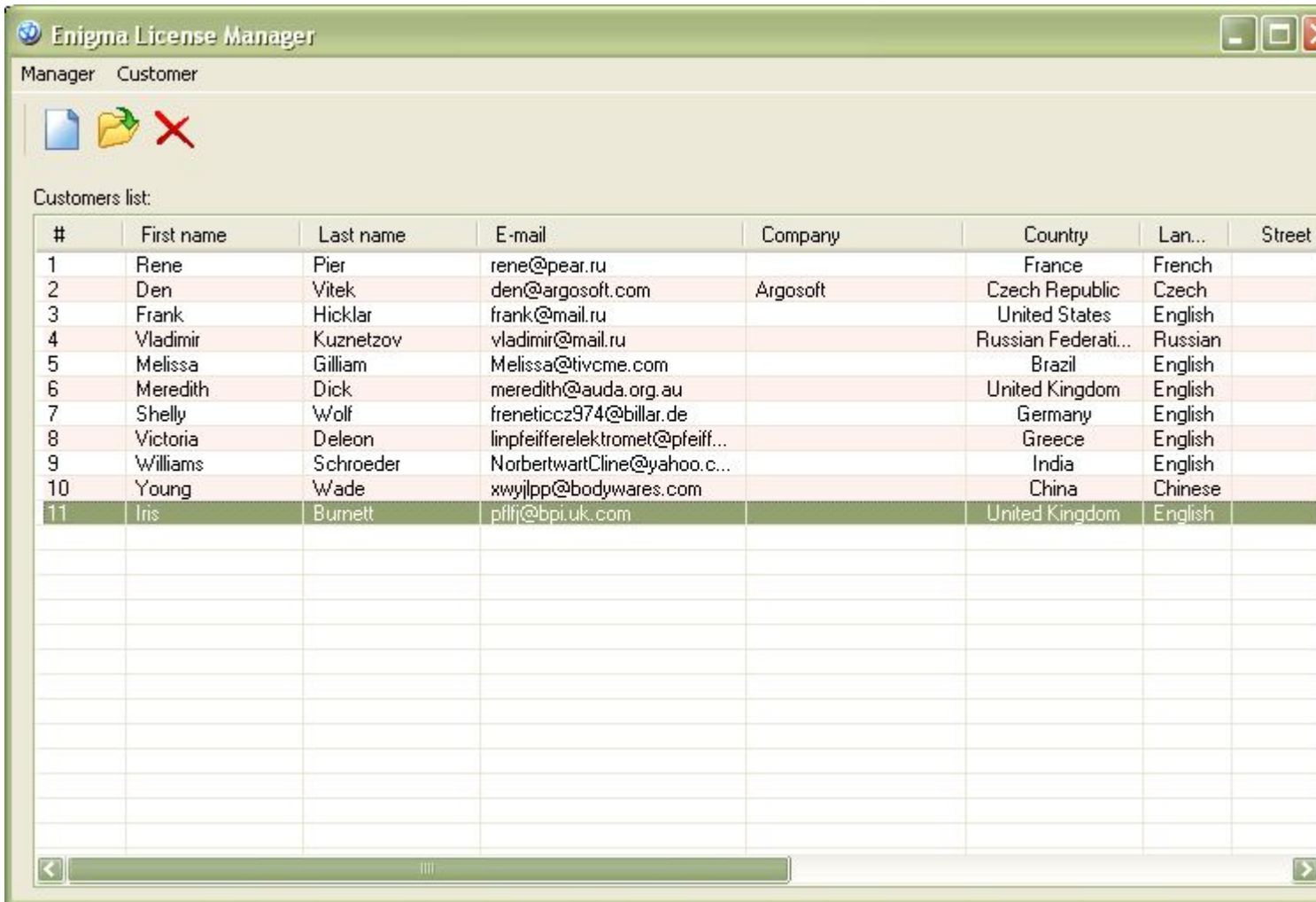
In the example, the Enigma API function `EP_TrialExecutions` is used to return the total and remaining number of executions. If the count of remaining executions is below half of the total trial executions, a warning message is displayed.

```
program test;
{$APPTYPE CONSOLE}
uses
  Windows,
  SysUtils,
  enigma_ide in 'include\enigma_ide.pas';
var
  Total, Left : word;
  sMessage : string;
begin
  // This is test application of trial Enigma API functions
  // If the function succeed then it returns true
  // If the function failed then
  // 1. the trial limit on executions number was not set before protection
  // 2. here is unexpected error
  if EP_TrialExecutions(Total, Left) then
  begin
    if Left = 0 then
    begin
      MessageBox(0, 'Your trial period has over!', 'Test application', 0);
    end else
    begin
      if (Total div Left) >= 2 then sMessage := 'You have left more then half executions of your trial! per
      sMessage := sMessage + format('You have left %d from %d days of your trial period!', [Left, Total]);
      MessageBox(0, PChar(sMessage), 'Test application', 0);
    end;
  end;
  // You may also use EP_TrialDays and EP_TrialExpirationDate
  // functions to control trial period
end.
```

Enigma License Manager

Enigma Protector License Manager is designed for storing and managing registered users information and licenses. The license manager allows adding full information about registered users and detailed info about user licenses. There are possibilities to mark registration keys as stolen (such registration keys will be blocked after protection), manage license subscription information (info about the subscribed license and user specified in Enigma Mailer) and generate licenses. In short, it is a powerful, useful and user-friendly manager for generating and storing license information. Click the Customer button to create a new customer record or select the necessary record and click the Edit Customer button to edit/view the record.

- [Adding/Editing Customer](#) (Main Menu - Customer - New/Edit)
- [Keys Generator Properties](#) (Main Menu - Manager - Key Generator Properties...)
- [Importing License Database](#) (Main Menu - Manager - Import Database...)



The screenshot shows the Enigma License Manager application window. The title bar reads "Enigma License Manager" and the menu bar shows "Manager" and "Customer". Below the menu bar are icons for file operations (New, Open, Save, Delete). The main area is titled "Customers list:" and contains a table with the following data:

#	First name	Last name	E-mail	Company	Country	Lan...	Street
1	Rene	Pier	rene@pear.ru		France	French	
2	Den	Vitek	den@argosoft.com	Argosoft	Czech Republic	Czech	
3	Frank	Hicklar	frank@mail.ru		United States	English	
4	Vladimir	Kuznetzov	vladimir@mail.ru		Russian Federati...	Russian	
5	Melissa	Gilliam	Melissa@tivcme.com		Brazil	English	
6	Meredith	Dick	meredith@auda.org.au		United Kingdom	English	
7	Shelly	Wolf	freneticcz974@billar.de		Germany	English	
8	Victoria	Deleon	linpfeifferelektromet@pfeiff...		Greece	English	
9	Williams	Schroeder	NorbertwartCline@yahoo.c...		India	English	
10	Young	Wade	xwyjpp@bodywares.com		China	Chinese	
11	Iris	Burnett	pflfj@bpi.uk.com		United Kingdom	English	

Edit Customer

Here you can enter information about registered users. All fields are optional, you may keep them blank or enter some values. Some fields can also be filled out in the [Enigma Mailer](#). Please, note also that the E-mail field is very important - this is the customer's email which will be used by the [Enigma Mailer](#) to send emails. In the right part of the window, there is a list of licenses associated with the current customer. Each customer may have any number of licenses (this is very useful if you use, for example, time-limited licenses). You may quickly generate/extend a license for an existing customer). To add/edit/delete a license click the following buttons.

- [Adding/Editing License](#)

The screenshot shows a software window titled "Edit Customer". On the left, there are several input fields for customer information: Company (empty), First name (Rene), Last name (Pier), Country (France), Language (French), Street address (empty), State/Province (empty), ZIP or postal code (empty), City (empty), Phone (empty), Fax (empty), and E-mail (rene@pear.ru). On the right, there is a table titled "Customer's Licenses" with columns: #, Stolen, Subscribed, Name, Key, and Hardware. The table contains three rows of data for Rene Pier. Below the table are "Add", "Edit", and "OK" buttons.

#	Stolen	Subscribed	Name	Key	Hardware
1	No	No	Rene Pier	XELKRT-AJEGY...	
2	No	No	Rene Pier	BLAWU6Y-TFX...	
3	No	Yes	Rene Pier	XELKQX-EZQ8...	

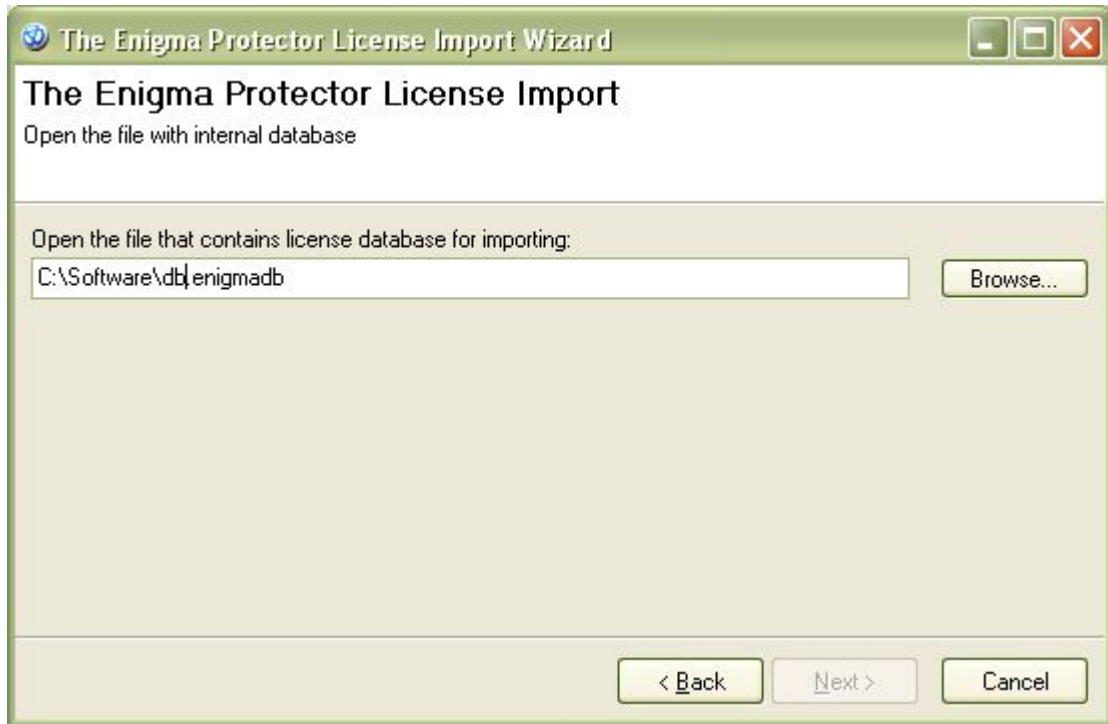
Additional Parameters - set up additional parameters for the customer record. To set up a list of customer additional parameters, click here to the [License Manager Properties](#). To learn how to access additional parameters through the Enigma Mailer, [click here](#).

Import Database

This feature allows importing License Manager databases from previously created projects into the current one and regenerate registration keys. It may help in the following cases:

- ┆ If you have created a new project for the existing application and want to update the users' registration keys
- ┆ If you have changed the parameters of the registration scheme (Hardware Locking type, Type or Base of registration keys)
- ┆ If you migrate a project from an older version of Enigma Protector to a newer one
- ┆ If you have decided to combine several projects

Step 1. Select the external database file name - files with .enigmadb extension. Click the Next button.



Step 2. Select the Import properties.

Re-generate registration keys - check it if you want to regenerate registration keys from the external database before adding to the current database. Please note that if registration keys stored in the external database are not valid for the current project and you uncheck this option, these keys will not be added.

Add hyphens to the keys - allows regenerating registration keys with hyphens.

Import expired keys - check to import expired keys.

Import stolen keys - check to import stolen keys.



Step 3. The import process has been started. Click the Finish button to exit the Wizard.



Edit License

Edit License window is very similar to the [Key Generator](#) window. It allows generating and verifying registration keys. In contrast to the simple interface of the Key Generator, the Edit License has a few additional fields. These fields are optional, and you may keep them blank or you may use them for the [Enigma Mailer](#). There you may generate or paste the existing registration keys, but note that ONLY valid registration keys will be added into the License Manager database.

The screenshot shows the 'Edit License' window with the following fields and options:

- Registration Name:** Rene Pier
- Registration Key:** BLFXU29-BRFT9ND-LV5JK9U-QH3U7EU (with 'Add Hyphens' checked and 'Copy/Paste' buttons)
- Hardware ID:** (empty field with a check icon)
- Key Expiration:** 1/ 1/2010 (checked)
- Select sections for decryption:** 16 checkboxes (Section 1-16) are all unchecked.
- Additional License Properties:** Number of Licenses: 0, License Price: 0, Reference Number: (empty), Stolen License: (unchecked)
- Subscription (Uses by Enigma Mailer):** Subscribed License: (checked), Expire Subscription: 1/ 1/2010 (checked), Generate Email: (button)
- Comment:** (empty text area)
- Buttons:** Generate, OK, Cancel

Registration Name - enter a registration name to generate a registration key for

Add Hyphens - if checked, the registration key is hyphenated

Registration Key - the generated registration key

Copy/Paste - click to copy/paste the registration key to/from clipboard

Hardware ID - enter the hardware ID to lock registration to a particular computer

 - click to check out the entered hardware ID

Key Expiration - enter the key expiration date

Number of License - enter the number of licenses that handle the current license

License Price - enter the price of the current license

Reference Number - the license reference number

Stolen License - if the current license (registration key) is stolen, check this box. After the protection is enabled, this key will be treated as invalid

Comment - any comment to the license

Options used in the [Enigma Mailer](#):

Subscribed License - if checked, an email will be generated for the current license in the Enigma Mailer after the Generate Emails button is clicked

Expire Subscription - enter the subscription expiration date. Emails will not be generated for the current license after the selected expiration date

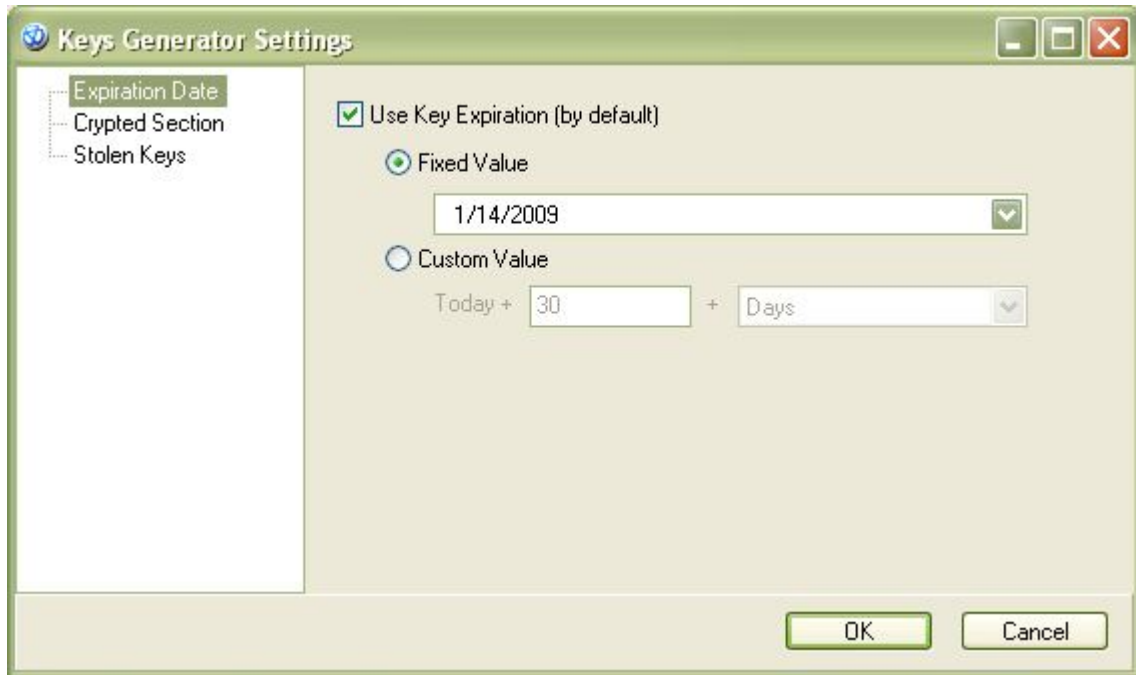
the latest used template. If you would like to associate several templates with this button, use the [License Manager Properties](#)

Additional Parameters - set up additional parameters for the license record. To learn how to set up the list of additional license parameters, refer to the [License Manager Properties](#). To learn how to access additional parameters through the Enigma Mailer, [click here](#).

Keys Generator Properties

Here you can set up the properties of the License Manager:

Expiration Date - set up default values for the Expiration Date field that will occur when the License Generator is opened.

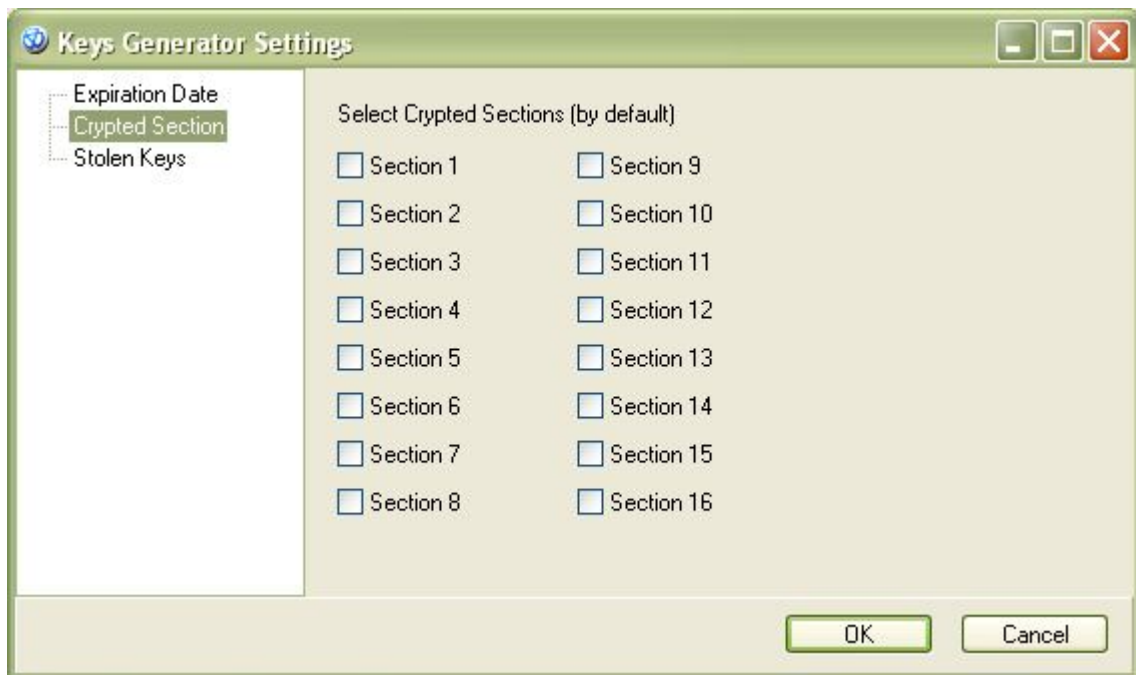


Use Key Expiration - indicates if the Expiration Date check box is checked when the License Generator is opened;

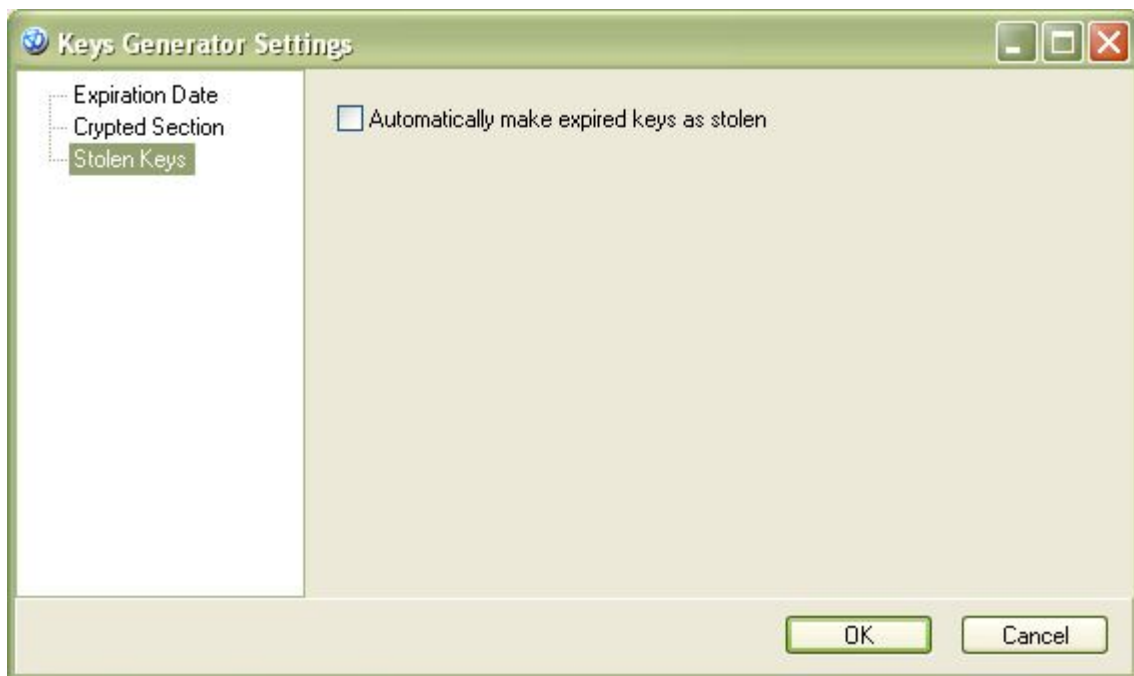
Fixed Date - the date the Expiration Date field will contain.

Custom Date - Expiration date is dynamic and depends on the current date.

Crypted Sections - set up default values for the Crypted Sections that will occur when the License Generator is opened.

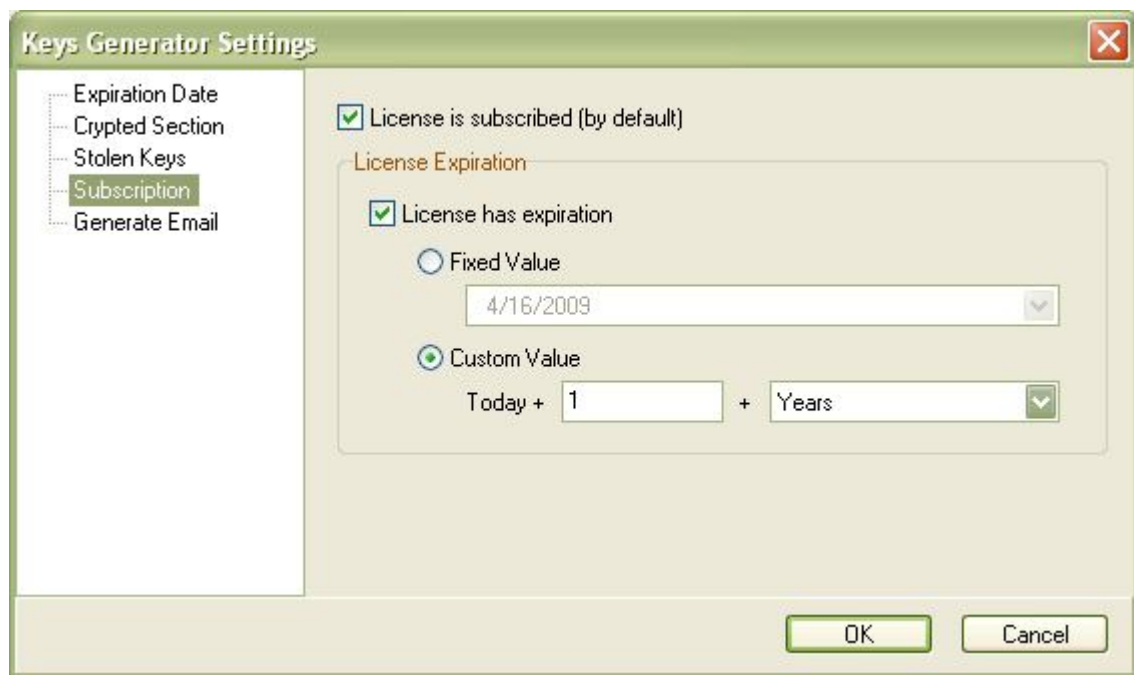


Stolen Keys

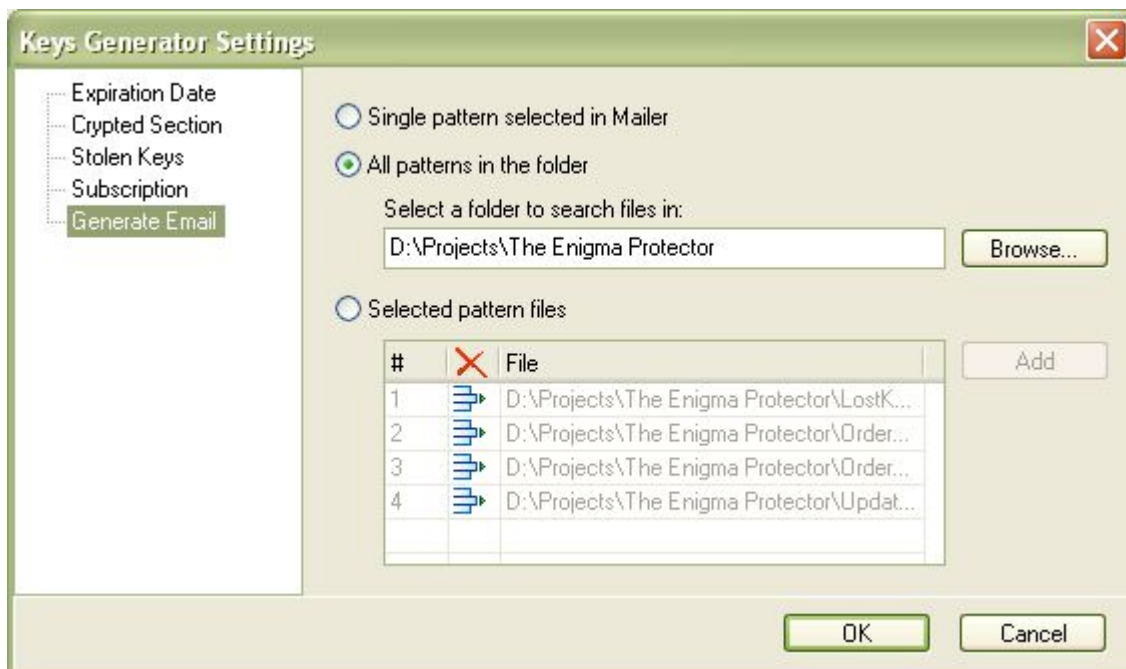


Automatically mark Expired Keys as Stolen - if checked, Enigma will automatically recognize expired keys as stolen with protection enabled. This will help avoid re-enabling of expired keys by manipulating the system clocks and add an additional protection level.

Subscription - enter subscription parameters. The License subscription check box and the subscription date will be filled out by default when the Edit License dialog is opened.



Generate Email - allows entering template files that will be associated with the Generate Email button in the Edit License dialog

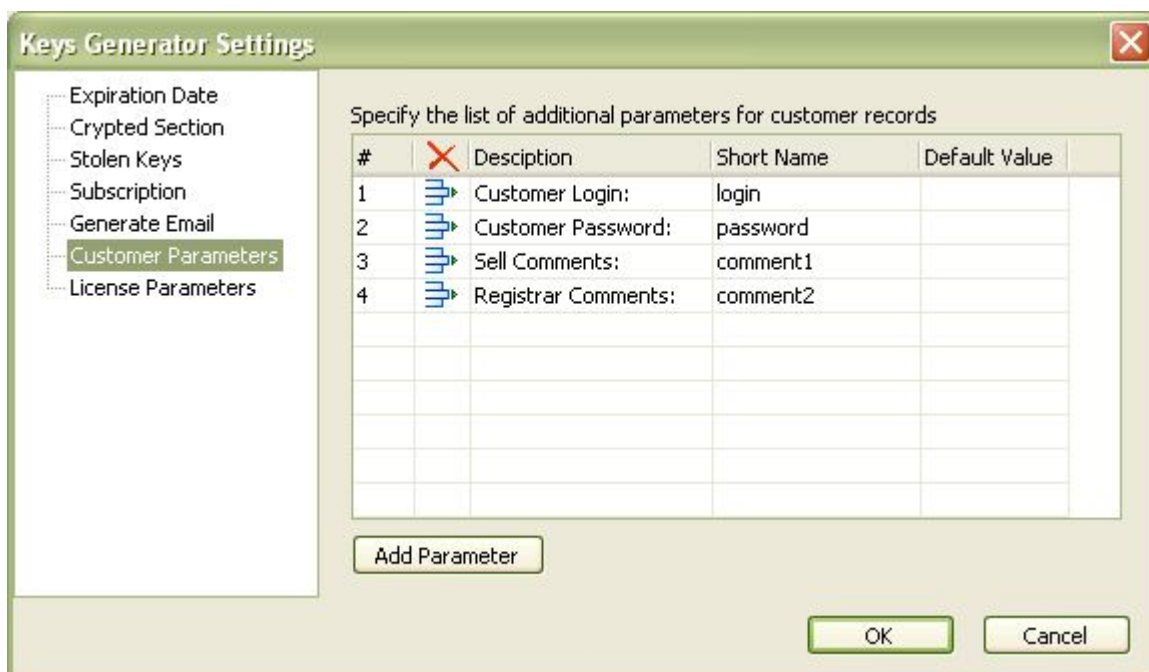


Single template selected in Mailer - the Generate Email button will be associated only with one template which was last used in the Enigma Mailer

All templates in the folder - select or enter a folder name. Enigma will analyse the folder and associate all the template files found in the folder with the Generate Email button

Selected template files - add template files to the list which will be associated with the Generate Email button

Customer Parameters - allows setting up an unlimited number of additional parameters for customer records. These parameters will also be stored in the License Manager database and can be available through the [Enigma Mailer](#). All parameters are of the string type.

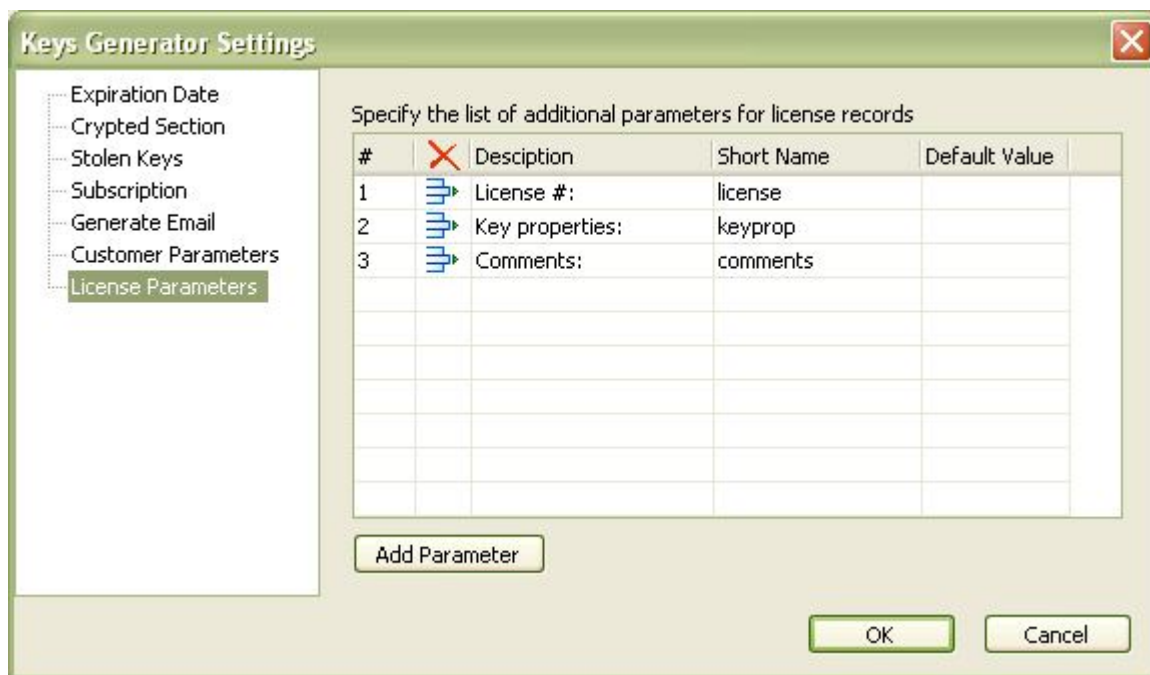


Description - enter the name or description of the parameter. This value will be shown in the Edit Parameters window. It is only used as a notice.

Short Name - the short name of the parameter. The parameter will be stored under this name in the License Manager database in the customer XML node. Also, this value is used to [access the parameter in Enigma Mailer](#) during messages generation.

Default Value - a default value for this parameter. Each newly created customer will have this parameter value by default

License Parameters - allows setting up an unlimited number of additional parameters for license records. These parameters will also be stored in the License Manager database and can be available through the [Enigma Mailer](#). All parameters are of the string type.



Description - enter the name or description of the parameter. This value will be shown in the Edit Parameters window. It is only used as a notice.

Short Name - the short name of the parameter. The parameter will be stored under this name in the License Manager database in the license XML node. Also, this value is used to [access the parameter in the Enigma Mailer](#) during messages generation.

Default Value - a default value for this parameter. Each newly created license will have this parameter value by default

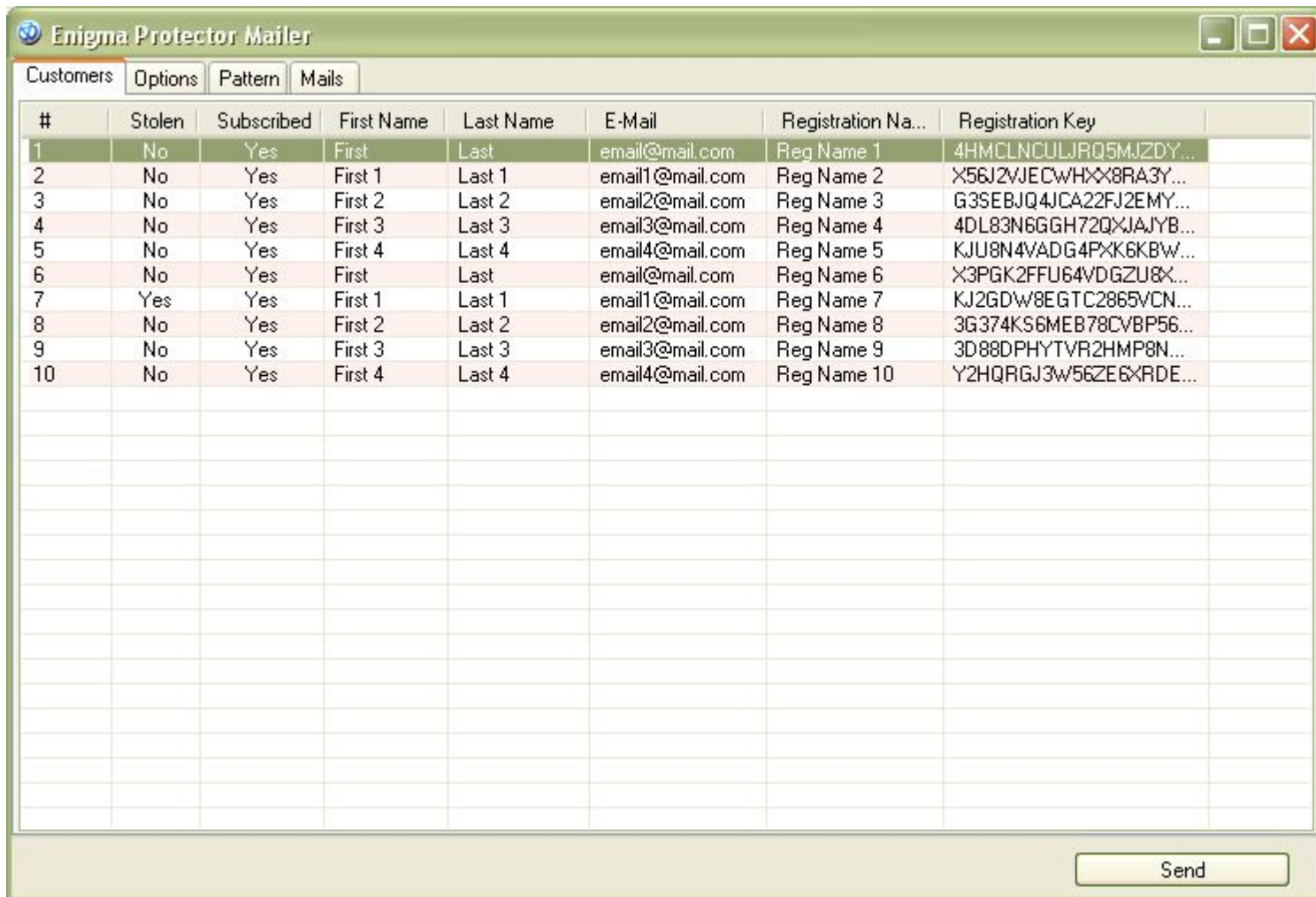
Enigma Mailer

Enigma Protector Mailer is a special tool, which allows generating and sending emails to registered customers. Every software needs to inform the registered users of a version update or personal user's registration information. It is long and difficult to create emails for each customer, and if you have several thousands of registered customers, it is simply impossible to manually generate emails for each of them. Enigma Mailer is the best tool to create and send email automatically, everything you need is just a registration database (which you can create with the Enigma Registration Manager) and a template. A template is a common text pattern that should be processed by Enigma Mailer and applied to the each customer.

- | [Customers List](#)
- | [Options](#)
- | [Template](#)
- | [Emails](#)

Customers List Panel

Customers List panel - all registration records from the database are displayed here. To create a database, use the [License Manager](#).



The screenshot shows the 'Enigma Protector Mailer' application window. It has a title bar with the application name and standard window controls. Below the title bar are four tabs: 'Customers' (selected), 'Options', 'Pattern', and 'Mails'. The main area contains a table with 10 columns: '#', 'Stolen', 'Subscribed', 'First Name', 'Last Name', 'E-Mail', 'Registration Na...', and 'Registration Key'. The table lists 10 customer records. At the bottom right of the window is a 'Send' button.

#	Stolen	Subscribed	First Name	Last Name	E-Mail	Registration Na...	Registration Key
1	No	Yes	First	Last	email@mail.com	Reg Name 1	4HMCLNCULJRQ5MJZDY...
2	No	Yes	First 1	Last 1	email1@mail.com	Reg Name 2	X56J2VJECWHXX8RA3Y...
3	No	Yes	First 2	Last 2	email2@mail.com	Reg Name 3	G3SEBJQ4JCA22FJ2EMY...
4	No	Yes	First 3	Last 3	email3@mail.com	Reg Name 4	4DL83N6GGH72QXAJYB...
5	No	Yes	First 4	Last 4	email4@mail.com	Reg Name 5	KJU8N4VADG4PXK6KBW...
6	No	Yes	First	Last	email@mail.com	Reg Name 6	X3PGK2FFU64VDGZU8X...
7	Yes	Yes	First 1	Last 1	email1@mail.com	Reg Name 7	KJ2GDW8EGTC2865VCN...
8	No	Yes	First 2	Last 2	email2@mail.com	Reg Name 8	3G374KS6MEB78CVBP56...
9	No	Yes	First 3	Last 3	email3@mail.com	Reg Name 9	3D88DPHYTVR2HMP8N...
10	No	Yes	First 4	Last 4	email4@mail.com	Reg Name 10	Y2HQRGJ3W56ZE6XRDE...

Select users from the list and right-click on them to generate emails for particular users.

Options Panel

Here are the common Enigma Mailer options used for connecting to the email server and sending emails.

The screenshot shows the 'Options' tab of the Enigma Mailer application. The 'EMail Address' field is set to 'mail@server.com'. Under 'Account Information', the 'User Name' is 'mail' and the 'Password' is 'password'. In the 'Mail Server' section, the 'SMTP Server' is 'smtp.server.com' and the 'Port Number' is '25'. The 'Use SSL' checkbox is unchecked. The 'Send Delay' section shows a delay of '3' minutes, with a note explaining that this value is used when the server response time exceeds a certain threshold.

Email Address

FROM Address - the email address from which the emails will be sent. Note: this address should not differ from User Name.

Account Information

User Name - the user name of the email account to connect to SMTP server.

Password - the password of the email account to connect to SMTP server.

Mail Server

You can obtain the SMTP Server name and Port Number from your email provider.

SMTP Server - the url of SMTP server to connect to;

Port Number - the port number for sending emails;

Use SSL - enables SSL connection to the mail server.

Send Delay

Delay - a delay in minutes in case a server error occurs. Sometimes the server generates an error while sending emails. lot of emails, the server replies that the number of outgoing emails exceeds the quota. If this occurs, Enigma Mailer will wait the number of minutes specified in the Delay field.

Template Panel

Here you can set up a template that will be used for automatic email generation.

The template consists of 4 parts (as an email itself): FROM, TO, SUBJECT and BODY fields. These fields can contain any regular expressions that will be replaced with the information from the users database. The table below shows regular the values they will be replaced with.

Regular Expression	Description
%CUSTOMERCOMPANY%	Customer's company name
%CUSTOMERFIRSTNAME%	Customer's first name
%CUSTOMERLASTNAME%	Customer's last name
%CUSTOMERSTREET%	Customer's street address
%CUSTOMERSTATE%	Customer's state/province
%CUSTOMERZIP%	Customer's ZIP or postal code
%CUSTOMERCITY%	Customer's city
%CUSTOMERCOUNTRY%	Customer's country
%CUSTOMERLANGUAGE%	Customer's language
%CUSTOMERPHONE%	Customer's phone
%CUSTOMERFAX%	Customer's fax
%CUSTOMEREMAIL%	Customer's email
%REGNAME%	Registration name
%REGKEY%	Registration key
%REGKEYCREATEDATE%	Registration key creation date
%REGKEYEXPIRATIONDATE%	Registration key expiration date
%REGKEYHARDWARE%	Registration key Hardware ID
%REGKEYLICNUM%	Registration key license number
%REGKEYPRICE%	Registration key price
%REGKEYREFERENCE%	Registration key reference number
%REGKEYCOMMENT%	Registration key comment

Access to additional parameters

To access additional customer parameters, quote the Short Name value of the parameter with %CUSTOMER at the beginning and % at the end. For example, if the Short Name of the additional customer parameter is "login", the regular expression used to access it in Enigma Mailer will be %CUSTOMERLOGIN%

To access additional license parameters, quote the Short Name value of the parameter with %REG at the beginning and % at the end. For example, if the Short Name of additional license parameter is "license", the regular expression used to access it in Enigma Mailer will be %REGLICENSE%



Enigma Mailer

Customers Options **Pattern** Mails

Load Pattern From File:

Pattern

"FROM" field: "CC" field (email address):

"TO" field: "BCC" field (email address):

"SUBJECT" field:

"BODY" field:

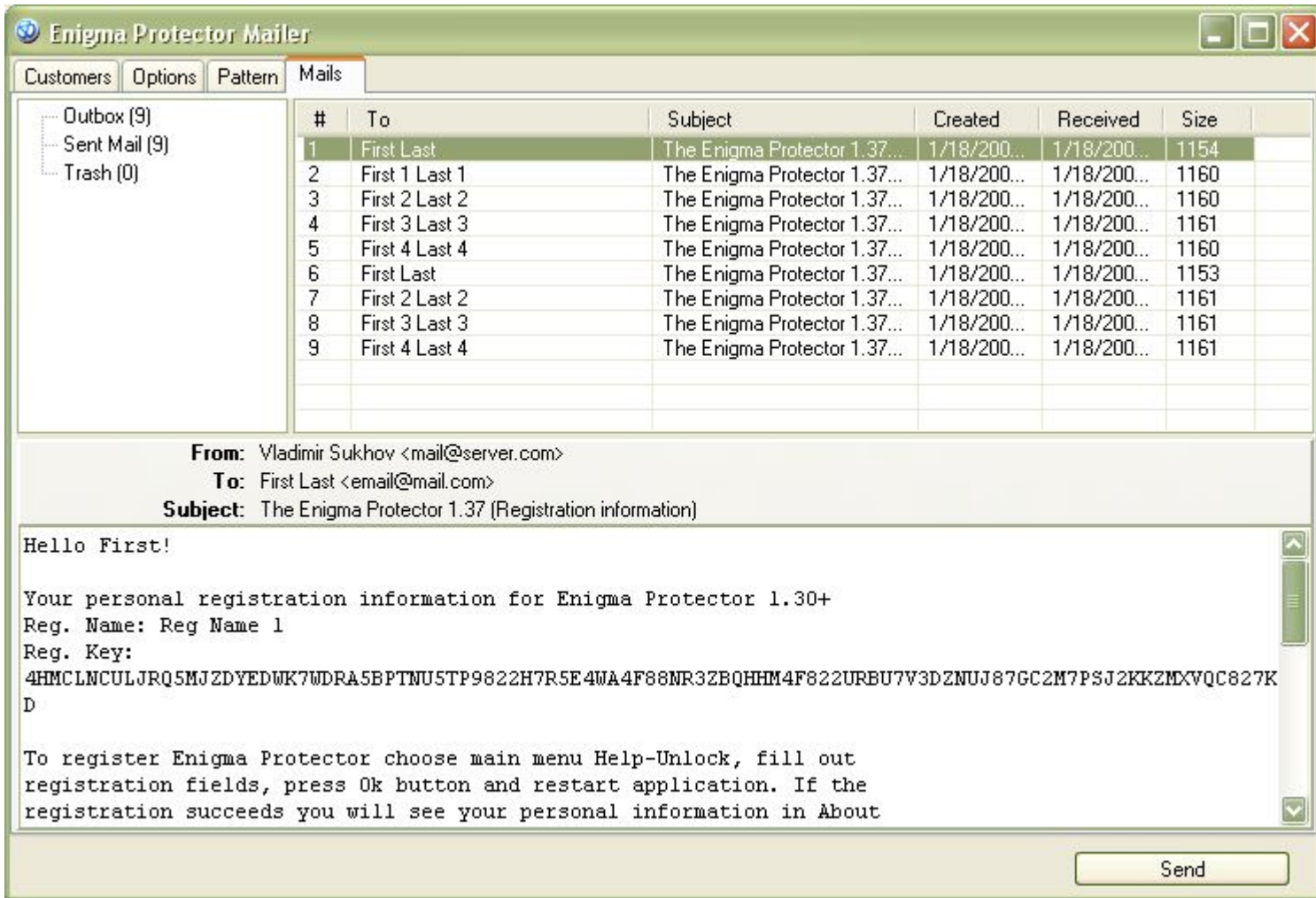
A template file is a simple xml file. You can find template examples in the "Email Patterns" folder of the Enigma Protector.

Generate Emails button - click it to automatically generate emails.

Note: the database from the Enigma Protector Registration Manager will be used for email generation. Emails will be generated only if the registration record is marked as "Subscribed" and not "Stolen". If the registration record contains a stolen flag, the emails will not be generated, irrespective of whether the record is subscribed.

Emails Panel

This panel displays the emails database. Outbox - emails prepared to be sent. Sent Mail - emails that have already been sent. Trash - deleted emails. To move emails from/to categories right-click your mouse and select the "Move to category" option. The "Delete popup" option deletes all emails in the selected category.



Send - click this button to send the prepared emails from the Outbox folder.

Bugs and limitations

There are the following limitations on The Enigma Protector usage:

1. it does not support modules without the entry point (these are dynamic link libraries that allow storing resources);
2. it does not support modules having anything like CRC checking;
3. it does not support multi-layered protection, i.e. using The Enigma Protector together with any other kind of packer/protector;

If you find any errors in The Enigma Protector or in the protected modules, please, let us know! While sending a malfunction report, describe all actions that led the module to the malfunction, as it is important for us to be able to help you. Also, we will be grateful if you could send us the project file and the unprotected version of your program module!

Extra Resource Protection

Use extra resource protection

Use extra resource protection - using additional features to protect the module resources. This implies compression and enciphering of all resource types (except the resources specified in the Compression-CompressionMode panel). The function allows you to hide the resource content from such tools as ResHack, Restorator, etc. If the option is disabled, all resources can be viewed.