**Proteus**

Nothing great in the world has ever

been accomplished without passion.

Hegel

Rel 3.00

# Table of content

|  |
| --- |
| [1. Table of content 4](#_Toc164601850)  [2. Proteus 6](#_Toc164601851)  [What is Proteus? 7](#_Toc164601852)  [Trial license 7](#_Toc164601853)  [Rent license (aka subscription) 7](#_Toc164601854)  [Permanent license 7](#_Toc164601855)  [Modules 7](#_Toc164601856)  [Free Proteus license for you 9](#_Toc164601857)  [What can Proteus do for you? 10](#_Toc164601858)  [Automates the process of selling your app. 10](#_Toc164601859)  [Prevents fraud 10](#_Toc164601860)  [HardwareID-based keys 11](#_Toc164601861)  [Proteus can even do marketing 11](#_Toc164601862)  [How to use it? 12](#_Toc164601863)  [Typical usage scenario 12](#_Toc164601864)  [Is it difficult to integrate Proteus into my app? 12](#_Toc164601865)  [Details 14](#_Toc164601866)  [Definitions 14](#_Toc164601867)  [Unlock key 16](#_Toc164601868)  [Setting up your app – Step by step guide 22](#_Toc164601869)  [Self protection 25](#_Toc164601870)  [Under the hood 25](#_Toc164601871)  [Anti-tampering system 25](#_Toc164601872)  [Stolen keys 25](#_Toc164601873)  [How to add a 'Stolen keys' list to your project? 26](#_Toc164601874)  [Amnesty key 26](#_Toc164601875)  [Certificate ID 27](#_Toc164601876)  [Entering the same key twice 27](#_Toc164601877)  [Active certificate 29](#_Toc164601878)  [Initialization 29](#_Toc164601879)  [Demo program 29](#_Toc164601880) |

# What is Proteus?

Proteus is a licensing system I designed to help us to convert a dull Delphi program into a commercial program that we can deliver to our customers as time-limited (trial) or feature-limited (shareware).

Multiple scenarios are possible:

## Trial license

The customer receives a trial version of your software. The trial could be fully functional or “crippled” - for example, it will not save the data to disk. Upon purchase, you send an unlock key to the customer that will switch the program from *trial* to *fully functional*. You don’t have to send an *key* to your customers, in this case.

## Rent license (aka subscription)

The customer can purchase from you a license that works for only x days. Later, the customer can upgrade to a permanent license.

## Permanent license

The program switches from trial to fully functional when the user purchases a license and receives an *unlock key* from you. A Permanent license never expires.

## Modules

You can send to the customers a *key* that unlocks all or only some of parts/modules of your program. Therefore, you can sell your program at various prices based on the modules purchased. For example, if you sell a game, the customers could pay $1 for each level they want to purchase. Proteus supports 16 modules.

# What can Proteus do for you?

## Automates the process of selling your app.

If we use an external payment processor, Proteus empowers us to fully automate the process of selling our programs. For example, I use BlueSnap.com as a payment processor. For products where I sell more than one license per week, I pre-generate 5000 unlock keys and upload them on BlueSnap.

BlueSnap and Proteus now take care of the rest. Upon purchase, each customer automatically receives from BlueSnap one of these *unlock keys*. The customer enters the key into the program. Proteus validates the key and unlocks the program.

So, I fully automated the whole process. I don’t even know who my customers are.

When all keys are used up (sold), I get a notification by BlueSnap, and I generate a new batch of keys and upload them.

## Prevents fraud

When a license key is leaked by a hacker on cracking websites, we can act by immediately invalidating that key. This is called a “stolen” key. When such a key is detected, Proteus can lock down the program so it cannot be used anymore. A legal message can be shown to the user to “encourage” him to purchase a valid license. For details, see the OnKeyStolen event.

Also, measures have been put in place to make sure the customer does not try to temper with the license key it received or with the installation date (for Trial versions).

## HardwareID-based keys

Proteus can be easily integrated with the Hardware ID Extractor library I created. By altering a single field in Proteus, the programmer can generate keys that work only in a specific computer (recognized by its hardware ID fingerprint).

## Proteus can even do marketing

Proteus is good for marketing also. With Proteus we can target groups of people (Paying customers/Trial users/Demo users/All users) and show “Call to arms” messages based on their group.

For example, we could show news and updates to already paying customers while showing discounts to people that haven’t purchased *yet* a license.

# How to use it?

## Typical usage scenario

Drag and drop a Proteus component onto your form. Deliver the application as a trial to your customers where it will work for 30 days (“Trial period”). When the time is up the program will switch to Demo mode.

In the Demo mode, we could cripple program's functionality severely, or you can show a "nag" screen to encourage the customer to purchase a license.

Once the customer pays for a license, we can send him a key to unlock the program. This key will switch the program from Demo mode to Purchased (fully functional) mode.

## Is it difficult to integrate Proteus into my app?

We will not see much code in this chapter because we only need one single line of code to integrate Proteus into our applications. Simply call “TProteus.Initialize” at application start-up. Crazy, right?

Here are the steps:

1. Install the Proteus package. Drop a TProteus control on your form.   
   Call “TProteus.Initialize” during application initialization.   
   Compile the delivered *Key Generator* and use it to generate a *key* (aka *certificate*) that will expire after x days. Insert this key into Proteus, in the TProteus.CurCertif property.

This certificate will make your app behave as a trial program: once the specified number of days has passed, you will be notified via an event (see below).

1. At your program startup, check the status of the currently installed certificate (key). If the certificate is in:
   1. Full mode: Let the user use your program normally.
   2. Demo mode: Disable the “save” menu and show a message to encourage the user to purchase a license.
2. Compile and deliver your application.

Proteus generates a TProteus.OnSwitchedToDemo event when the trial period expires. Write a handler for that event and decide what to do. Well, not even that. For lazy people, Proteus already offers predefined messages for those event handlers, such as:

* ShowSwitchToDemo,
* ShowKeyAccepted,
* ShowKeyUnknown,
* SwitchToDemo,
* SwitchToDemoFallBack.

Ideas: In the OnSwitchedToDemo event handler we could announce to the user that the trial has expired, we could redirect it to the Purchase webpage, we could shut down the program, we could cripple the program (allow all function except “Save work”). You are limited only by your imagination.

A full-blown demo app is also available.

# Details

## Definitions

### Certificate

A certificate is a record that informs your program about the status of the license. It contains info such as:

* Program expiration date
* Current program state: trial/demo/full
* Modules unlocked (modules purchased and available for use)
* Customer related data (customer name, organization)
* etc

The certificate can be:

* Permanent (it unlocks the program forever)
* Temporary (it unlocks the program for a predetermined period)

A program can have multiple certificates installed in its storage area, but only one can be active. This one is called the *current certificate,* and it is available via the *TProteus. CurCertif* property. The programmer cannot/should not switch between certificates. Instead, Proteus itself implements the logic to choose the appropriate certificate. For example, an application was delivered with a default “Trial certificate”. Upon purchase of a license, a key is sent to the user. When the user enters that key into the program, a new certificate (called “Full license”) appears in the system. Proteus will choose the better (the “Full license”) certificate. The lesser certificate (the “Trial”) will be ignored until the current certificate expires (if ever).

Also, if the user was running on a temporary license, like “Subscription 30 days”, after 30 days, upon certificate expiration the program will switch from this certificate to another certificate IF available. For example, if the user still has time left on his Trial certificate, Proteus will switch to that certificate and run until whatever time was left on that Trai license, is used up. If the Trial certificate has no valid time left on it, the program will switch to Demo mode.

Here is its declaration. Note that some fields and methods (especially those for logging and debugging) were removed because they are less relevant. The comments are a bit standard because of the formatting of this book, and are under the fields/methods.

RCertificate= record

ID : String; { Unique ID for each certificate type. I use this ID to prevent the user from entering the same key twice (for example a trial key, or a key that works for only x days) }

CertifType : TCertificateType; { Registered or not: ctUninstalled, lsDemo, lsTrial, ctTemporar, ctFull }

{ User/license details }

UserName : String; { Can also be used for company name if separated with a proper separator }

OrgType : TOrganizationType; { In which type of organization the user works. Ex: Company/Academia/Home }

Edition : Byte; { Ex: Lite/Extreme/Ultimate Network/Single/Site }

NoOfLic : Byte;

{ Product }

ProductName : String; { The product/application for which the certificate is generated. Let's say you want to protect an application called 'My Great Program'. Put the TProteus control on your app's main form and set the ProductName field to 'Hello World'. Now start the KeyGenerator demo app and in the "Application name" enter also 'Hello World'. Now press the "Generate key" button. An unlock key will be generated and it will only work with an application called "Hello World". Please note that that this is not related to Application.Title. }

ProductVersion : Byte; { Key works with this (major) version only }

UnlockDown : Boolean; { If TRUE, the key works with this version AND all versions below }

{ Limited license }

isTrial : Boolean;

UpgradeTrialKey: Boolean; { MAKES SENSE only for 'Trial' certificates. If the user is in the trial (or demo) period and he downloads a new trial version and this new trial version has a new key, upgrade to it (get a new full trial period) }

FallBackToTrial: Boolean; { MAKES SENSE only for 'Limited' certificates. When a Rent (limited) key expires, allow the user to fall back to a fully functional trial (if there is an existing expired trial certificate, it will be reset and work again all 30days) }

ShowRemainTime : Boolean; { Show/hide remaining time (count down) for temporary keys. I might want to show it for rent keys, and hide it for temporary keys (keys that I send before I send the Final key, while awaiting payment) }

{ License exp }

KeyUseBefore : TDateTime; { The key must be entered before this date. After this it won't work }

KeyExpDate : TDateTime; { The key will be valid until this date (excluding) }

KeyValidMin : Cardinal; { The key will work for x minutes from the moment it was entered. KeyExpires and KeyValidMin cannot be used simultaneously}

KeyGenerated : TDate; { Date when the key was generated } { It is important that this field only stores the date, otherwise, if it stores also the time, the key generated will be different each time i press the 'Generate key' button because the 'second' portion of Now will change! }

{ RUNTIME FIELDS }

LastSeen : TDateTime; { Each time the program starts ups/ shuts down it updates this value. If in TRIAL\_MODE and the current system clock is smaller than this value then the user has tempered with system's clock. }

private

Initialized : Boolean; { Only for debugging }

FInstalled : TDateTime; { This field will be generated by the application (not by the issuer of the key) when the key is first installed. }

ModulesMask : Word; { Mask of bits (stored as Word) that shows which modules are unlocked by this certificate. Since this is a Word, and each bit encodes a module, we can have up to 16 modules (0..15). To activate the first module use '10000000 00000000'. To activate first and last module use '00000000 00000001'}

public

{ SERIALIZATION }

function GenerateKeyString: String; { Converts certificate data to a human readable key/string }

function DecodeKey(KeyString: String; UserEnteredKey: Boolean= FALSE): Boolean;

{ LICENSE STATE }

function PlatitFull: Boolean; { Fully paid }

function Platit: Boolean; { Fully paid / active rent }

function Running: Boolean; { Paid / active rent / active trial }

function CountingDown: Boolean; { Active trial / active rent }

function Trial : Boolean; { Active trial }

function Demo : Boolean; { Demo / Uninstalled }

procedure ModuleEnable (ModuleNumber: Byte);

procedure ModuleToggle (ModuleNumber: Byte; Enabled: Boolean);

function ModuleEnabled(ModuleNumber: Byte): Boolean;

{ LICENSE STATE }

function Stollen(StolenKeys: TStringList): Boolean; { Returns true if this certificate is listed in the 'stolen keys' list }

function InTheFuture: Boolean;

function OrganizationType: string;

function TimeLeft: Int64;

end;

## Unlock key

The *unlock key* (or simply “the key”), is a text representation of a certificate (see “certificate” definition above).

Simply put, the certificate and the unlock key are one and the same thing. The small difference is that the certificate is a binary Delphi structure while the key is a human readable string (so it can be sent to the customers by email). We will use the two terms interchangeably in this document.

You can also view the text key as a “recipient” or “transporter” of the binary certificate.

Proteus generates the text ley by passing certificate’s binary data through a MIME encoder and to get a human-readable string. Don’t worry, the binary data is first encrypted, and a checksum is also inserted into the stream. So, the certificate cannot be easily tempered with. Proteus also allows you to call your own encryption algorithm if you want.

Upon purchasing a license, the user receives a key from you. The user enters the key into the program. You can use the existing TProteus.EnterKey method to display the GUI where the user can enter his key, or you can define your own GUI.

Proteus validates the key entered by user. If the key is ok, the program is unlocked. If not, a friendly error msg is returned to the user asking the user to enter the key again.

### Certificate serialization

The certificate has a method called *serialize* that is converting its binary data to an array of bytes which later is encrypted and converted to a string:

{ Get certificate data serialized (as string). The CRC is NOT included. }

function RCertificate.serialize: TBytesArray;

VAR Stream: TCubicMemStream;

begin

Stream:= TCubicMemStream.Create;

TRY

{Magic number}

Stream.WriteByte (ctMagicNumber);

Stream.WriteByte (CertificateVer);

{ Certificate }

Stream.WriteStringA (AnsiString(ID));

Stream.WriteByte (Ord(CertifType));

{ User/license details }

Stream.WriteStringA (AnsiString(UserName));

Stream.WriteByte (Ord(OrgType));

Stream.WriteByte (Edition);

Stream.WriteByte (NoOfLic);

Stream.WriteWord (ModulesMask);

{ Product }

Stream.WriteStringA (AnsiString(ProductName));

Stream.WriteByte (ProductVersion);

Stream.WriteBoolean (UnlockDown);

{ Limited license }

Stream.WriteBoolean (IsTrial);

Stream.WriteBoolean (UpgradeTrialKey);

Stream.WriteBoolean (FallBackToTrial);

Stream.WriteBoolean (ShowRemainTime);

{ License exp }

Stream.WriteDate (KeyUseBefore);

Stream.WriteDate (KeyExpDate);

Stream.WriteCardinal(KeyValidMin);

Stream.WriteDate (KeyGenerated);

{ RUNTIME FIELDS }

Stream.WriteDate (Installed);

Stream.WriteDate (LastSeen);

Stream.WritePadding (8);

Result:= Stream.AsBytes;

FINALLY

FreeAndNil(Stream);

END;

end;

The opposite of this process happens in the *DecodeKey* which in the end calls *deserialize*.

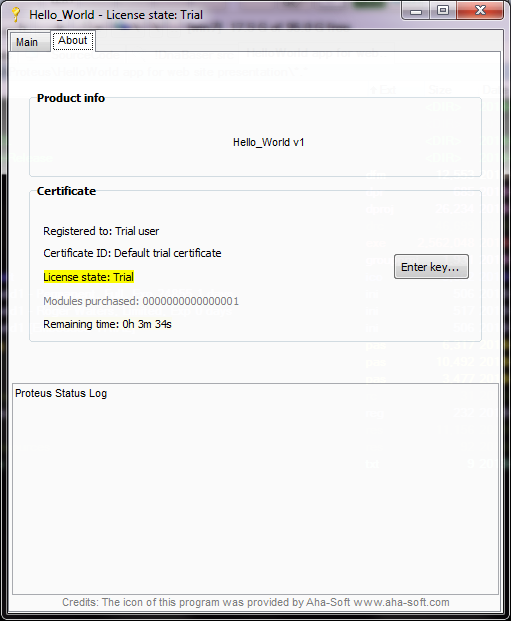
### Trial certificate

This is a special certificate that allows the program to work for a limited period, then switch to Demo mode. You can set this period to *any* value between 1 minute and 100 years.

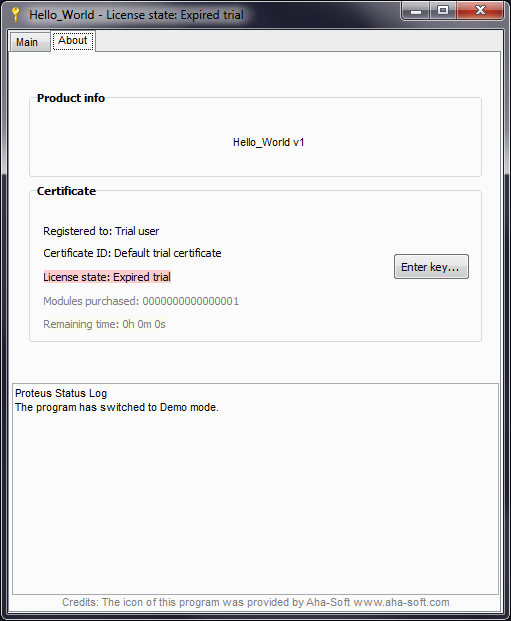
You can program your app in such a way that:

* In Trial mode, all or only some of the program’s modules are available.
* In Demo mode, the program's functionality can be severely limited. For example, the 'Save' button does not work anymore so the user cannot save its work until it purchased a license.

Even more flexibility: If the user is in the trial period and he downloads a new trial version (update) and this new trial version has a new trial key, you can decide if Proteus will use the new Trail certificate (get a new full trial period) or stick to wherever time was left onto the current certificate. For this just set the *UpgradeToNewTrialKey* field to True.



A program in its Trial period (3 minutes left).



The program after the trial period expired.

### Default key

Any program must be delivered with a default certificate.

Usually, this will be a Trial certificate, but it can be any certificate you can concoct.

### Dealing with program updates

We can choose if a key will unlock only the current version or also other versions, older or newer. For example:

* When a minor update is available (for example v2.0 to v2.1), the customer can download an update from your website. If the customer already has a valid certificate installed on his computer (Full or Rent), upon installing the update, the newly downloaded version will automatically use the current certificate.
* When a major update is available (for example v2 to v3) the programmer can specify how Proteus should behave: allow the user to use the existing license for v2 in v3 or not (the user has to purchase a new license in this case).

### Modules

You can compartmentalize your program into "modules" - blocks of code that perform certain functions. For example, if your program converts images, one module could be the BMP to JPG converter, and the other module could be the BMP to PNG converter. The user can choose to purchase only one module or both.

The key you will send to that customer will unlock only the modules purchased.

Proteus allows up to 16 modules. The modules are stored in the RCertificate.ModulesMask field:

{ Mask of bits (stored as Word) that shows which modules are unlocked by this certificate. Since this is a Word, and each bit encodes a module, we can have up to 16 modules (0..15). To activate first module use '10000000 00000000'. To activate first and last module use '00000000 00000001' }

ModulesMask: Word;

cpProteusUtils.pas offers the tools you need to set/clear a specific bit in the ModulesMask word.

After we have seen how cool, simple, safe, complete, yet flexible Proteus is, let me brag: Can you get licensing system more flexible than Proteus? 😊

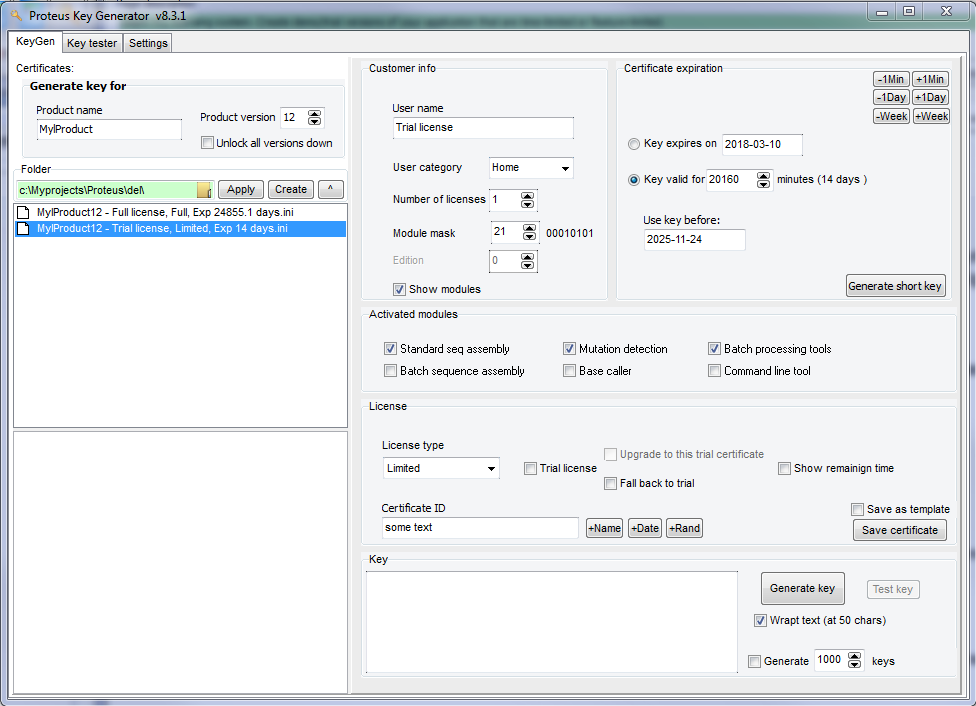
# Setting up your app – Step by step guide

To start delivering and selling your program to your customers, you need two things:

* 1. A cool app delivered as Trial (that expires after x days)
  2. A key generator so you can generate keys for your customers.

Below is an example of a complete Key Generator app, tailored for my personal needs. As you can see in the screenshot of this key generator, Proteus really offers you a lot of functionality and flexibility.

Of course, you can easily build your own key generator, tailored for your needs. You can start with the demo key generator provided with Proteus. Yours does not necessarily have to be this complex. Let’s build you a basic key generator. How hard can that be?



### Building the Key Generator

1. Start a new Delphi VCL app.

2. Put a button on is and call this function:

function GenerateKey: string;

VAR CurCertif: RCertificate;

begin

CurCertif.Reset; { We initialize this certificate to some default values}

{ Certificate }

CurCertif.ID:= 'Enter any random text here';

CurCertif.CertifType:= ctCountDownActive; { Count down Trial period }

{ Limited license }

CurCertif.IsTrial:= True; { Works hand in hand with ctCountDownActive }

{ Product }

CurCertif.ProductName:= 'My cool app'; { Name of the product we want to unlock }

CurCertif.ProductVersion:= '1.0.0.0'; { Key works with this version and up }

{ Key }

if radKeyExpirationType.Checked

then CurCertif.KeyExpDate := StrToDate('2024.01.01') { Expire at this date }

else CurCertif.KeyValidMin:= 60; { or, Expire after 60 min }

{ Obtain the key-string }

Result:= CurCertif.GenerateKeyString;

end;

In the code above, we initialize a certificate and we set it to be a Trial certificate that will count down from sixty minutes to zero. In the end, the string returned by this simple *GenerateKey* function is a Trial key. Pretty straight forward, right? Now you have a simple key generator.

You will enter the key in the Proteus component that is protecting your shareware app, in the *TProteus.DefaultKey* property.

Let see how to integrate Proteus in your shareware app:

### Preparing your application

Once your application is ready to be delivered to the customers:

* Drop a TProteus control on your main form.
* Set the 'ProductName' *to the same* value you used in the *GenerateKey* function above.
* Set the 'AutoStoreCertificate' to True to automatically save the certificate to the storage area.
* Set the 'ObfuscateRegistry' to True for higher protection or to False for easier debugging.
* Enter the certificate generated with the *GenerateKey* function above, in the 'TProteus.DefaultKey' field.
* Add some in the 'OnSwitchToDemo' event handler.
* If you offer a time-based trial:
  + Call 'CheckComputerClock' from time to time to make sure the user didn't set the system clock back.
  + Use the ShowRemainingTime function to display what's left from the trial period.
* Finally, call “Proteus.Initialize” at your app startup.

Your application is now a true Trialware application. Ship it to your customers. Ask a lot of money for it 😊. The moment the customer runs the app, it will start to count down. After 60 minutes, it will switch to Demo mode.

# Self protection

## Under the hood

At startup, Proteus checks if any certificate is already installed. There could be multiple certificates installed, but only one is the active/current certificate.

If a certificate is found, it is decoded and loaded into the 'CurCertif' field. Then Proteus sets the LicState field (demo, trial, full) according to the current certificate.

If no certificate is found, Proteus uses the default certificate embedded within the program. Usually, this is a “Trial” certificate.

## Anti-tampering system

At startup, the program also checks if the system date has been tampered with. The *TProteus.LastSeen* field is used for this. Proteus can detect if the user tries to reset the Trial period by setting back the computer’s clock. An event is triggered by Proteus in this case. You can write code to decide what you do in this case (lock down the program, etc).

## Stolen keys

StolenKeys is a resource file that is compiled into the program. If one of the keys that we sent to our customers was leaked on serials/cracks website, we can put it in this list, recompile the program and release an update.

Proteus checks the certificate installed in user’s computer against the “StolenKeys” list at each startup. If the certificate is found in the Stolen Keys list, Proteus will permanently switch to Demo mode. The user is not allowed to enter any more keys, even if he has a valid key. This way we prevent this fraudulent user from trying to use more stolen keys.

## How to add a 'Stolen keys' list to your project?

This step is optional, but we can do it for fun, to see how it works.

Generate one key. We would pretend that this the key was “leaked”.

1. Put the leaked key in 'CheiFurate.txt' using the following format:

CertificateID=UserName

Be careful not to use spaces around the '=' sign.

If you have multiple keys, put one key per line.

2. Create a new file called 'ResurseIncluse.rc' and write this inside:

Blacklist RCDATA CheiFurate.txt

3. Bind the RC into your app (drag and drop it into the Project Manager):

{$R 'ResurseIncluse.res' 'ResurseIncluse.rc'}

Compile and run the program. Now if we try to enter that key into the program, Proteus will recognize it as a “stolen key” and refuse it.

## Amnesty key

This allows a program that was forcefully put in Demo mode (because of a stolen key), to receive a valid key again.

To do so, the user will have to enter a “secret” keyword as a key. You will find the keyword in the source code, in the "ShowEnterKeyBox" function.

# Certificate ID

The Certificate ID is a random text (see *RCertificate.CertificateID* field) added to the certificate. In most cases it is recommended to use a unique text for each certificate you generate. For example, if you send a key to a customer that purchased a license, we might want to put a unique signature (Certificate ID) in that key so we will know from whom the key was generated, in cases the key was leaked over the Internet (or shared with friends/coworkers).

Another usage for a unique ID is when you generate multiple licenses for the same customer. In this case each key must have a unique ID so the program will know which one was used up (expired) and which one is new.

However, for special certificates, we want the ID to be non-unique. This is the case with Trial certificates. The Trial certificate is the same for all users so the ID for such a trail certificate is always the same. This prevents the user from getting a new Trial period when he downloads an update.

## Entering the same key twice

Proteus has an internal check, based on CertificateID, that prevents the user from entering the same key twice. This prevents a user from reusing a time-limited key repeatedly.

When the user tries to enter a key, Proteus checks if a certificate with the same ID already exists in the system. If it does, the newly entered key is refused. This influences the behavior of Proteus only if that certificate is a time-limited certificate in the following way:

#### Trial certificates:

* When an update is released (for example, v1.2 to v1.3), the CertificateID delivered with this new version should remain the same as the one of the previous versions, to prevent the user from re-starting the trial period.
* When an upgrade is released (for example, v1.3 to v2.0), the CertificateID of the new version could be changed to allow the user to trial this brand-new version.
* When we to change the running time of a Trial key, the programmer can change the certificate ID let the user start a new trial period or could let the certificate ID the same, case in which the running conditions (running time) of the old key will remain the same, and the user will use up whatever interval he has left from the previous Trial certificate.

#### Rent (subscription) certificates:

* When a second (or third, etc) **Rent** key is generated and sent to the customer, the CertificateID field should be different for each new key, otherwise Proteus will refuse the new key, thinking that the second key is the same as the previous key.

The Full certificates are not affected by the above algorithms.

Personally, in my key generator, when I generate keys, I append the current date & time to the CertificateID field, to make sure I always get a unique ID.

# Active certificate

If multiple certificates are found in the storage area, ONLY one certificate can be active.

The last certificate entered becomes active and inactivates all other active certificates.

In the case of Trial or Rent certificates, when the active certificate expires, the program switches to DEMO mode, but that certificate still remains the Active certificate. It will remain like this until a new certificate is entered.

An exception to this is when the programmer set the *FallBackToTrial* field to True, in which case Proteus would switch back to the default Trial certificate (the one delivered with the program) and use whatever time was left on that certificate (if any).

## Initialization

Proteus should be initialized at program startup by calling *TProteus.Initialize*.

If during startup, Proteus shows the "Trial expired" message box, the application will halt until the user presses *ok*.

If we want to allow the program to continue loading even if the "Trial expired” message box is shown, we could use a non-blocking message box, like the “FromAsyncMessage” procedure in LightSaber library. However, it is recommended that the program is paused until the user acknowledges that he understands that the program is running now in Demo (limited) mode.

## Demo program

A Proteus demo program is available for download. Extra documentation can also be found in the source code - all functions are fully documented.

Details about Delphi Proteus License Manager are available at [www.GabrielMoraru.com](http://www.GabrielMoraru.com)