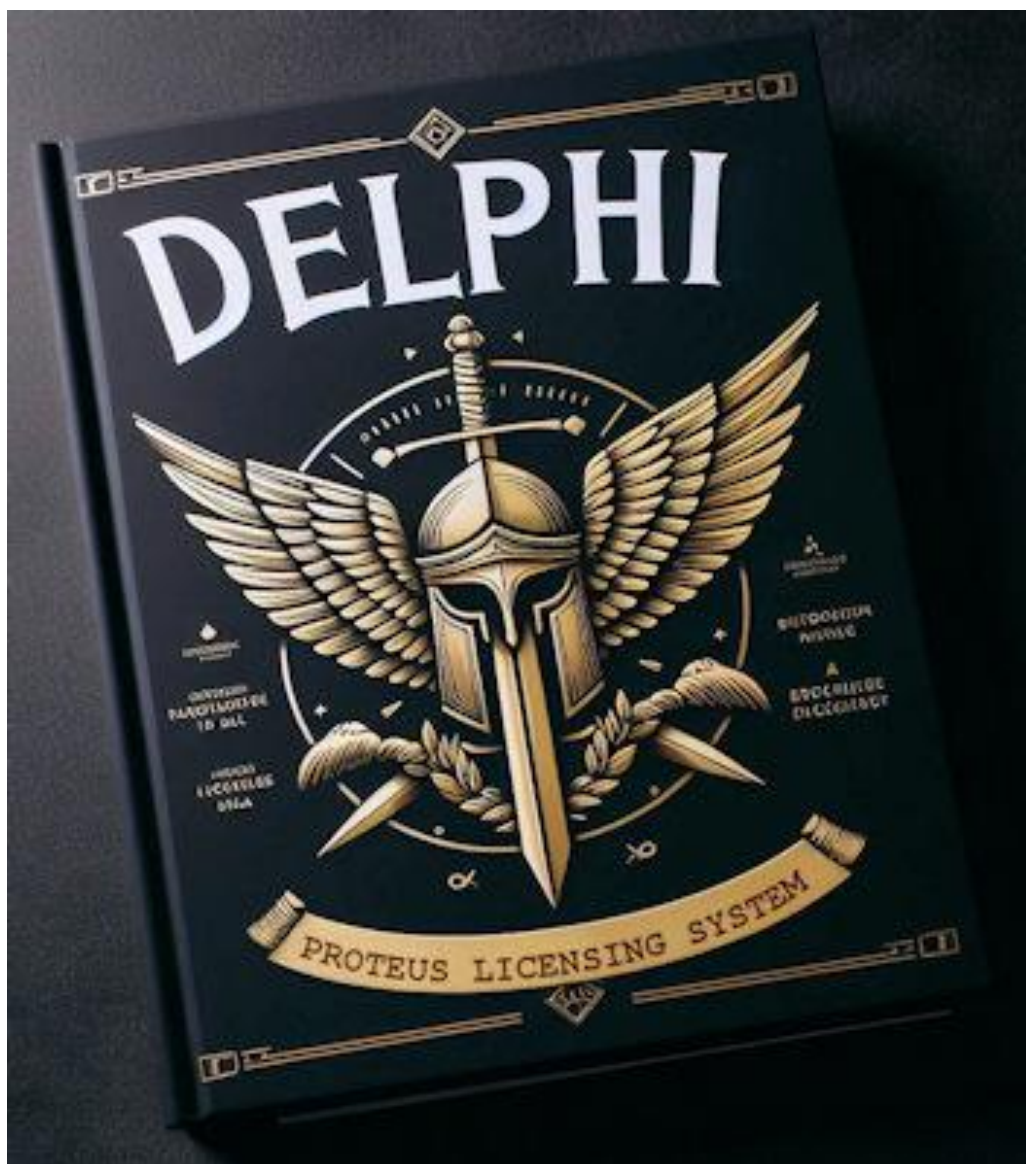


Proteus



Release 2.70

“Modularization isn't about breaking code into pieces.
It's about composing a masterpiece
from harmonious parts.”

1. Table of content

1. Table of content	3
Symbols and conventions	5
2. The Proteus licensing system	6
What is Proteus?	6
What can Proteus do for us?.....	6
Usage scenarios.....	8
Modules.....	9
Is it difficult to integrate Proteus into an existing program?.....	9
Details	11
Library structure.....	11
Definitions	12
Code structure.....	20
Self-protection	24
Certificate ID.....	26
Handling multiple certificates	28
The “Trial expired” message.....	29
OnSwitchToDemo	29
The Key Generator	30
Documentation	32
Online activation server.....	33
How does it work?.....	33

My books	35
Already published books	35
Next books	35
Contact me!	36

Symbols and conventions

The following symbols and conventions are used in this document:

Warning



Take great care when this symbol is encountered while discussing a topic.
Don't take that warning lightly.

Source code

The code or file paths are enclosed in a cassette like this:

```
Procedure CodeExample;  
begin  
end;
```

Code formatting style

In most cases I respect the default Embarcadero code style. The main exception is if/then/else which I write it as:

```
if x > 0  
then DoSomething1  
else DoSomething2;
```

instead of:

```
if x > 0 then  
    DoSomething1  
else DoSomething2;
```

2. The Proteus licensing system

What is Proteus?

In order to sell a software product, we need effective licensing strategies. Enter Proteus—a flexible licensing system that helps us convert Delphi programs into commercially viable products that can be delivered to our customers as time-limited or feature-limited trials.

No solutions for Delphi?

If you look up the Internet, you will find a few licensing systems (usually available as DLLs), but none of them is dedicated to Delphi.

Proteus emerged from my necessity to retrofit my own commercial products (such as DNA Baser Sequence Assembler) with a Delphi-native licensing system. Today Proteus is a mature product that safeguards the intellectual property of numerous Delphi companies/products.

What can Proteus do for us?

Unsupervised sales

Combined with an external payment processor such as PayPro.com, Proteus empowers you to fully automate the process of selling your programs. For products where I sell more than one license per week, I pre-generate 5000 *unlock keys* and upload them on PayPro.

PayPro and Proteus now take care of the rest: upon purchase, each customer automatically receives from PayPro one of those unlock keys. The customer

enters the key into the program. Proteus validates the key and unlocks the program.

This way, 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 PayPro, and I generate a new batch of keys and upload them.

Prevents fraud

When a license key is leaked (this is called in Proteus a "stolen" key) by a customer and it appears on cracking websites, we can act by invalidating that key.

When Proteus detects such a key it locks down the program so it cannot be used anymore. The programmer can show a message to the customer to "encourage" him to purchase a valid license. For details, see the OnKeyStolen event.

Also, Proteus has measures put in place to lock the program if a customer/hacker tries to temper with the license key.

Hardware ID-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.

Usage scenarios

Proteus is truly flexible, allowing for multiple *delivery scenarios*, such as:

Trial license

No respectable company delivers a product without letting its customers “taste” it first. In today’s world, if you don’t offer a trial, it means that you have something to hide (your product is not good).

Use Proteus to “bake” a trial version of your product. The trial could be:

* *Fully functional*

All features will work, but only for a period of time (usually 14 or 30 days).

* *Crippled*

The program will work for unlimited time, but some features will not be available. For example, it will not save the data to disk, or it will add a watermark over the saved data.

Upon purchase, the customer receives an unlock key that will switch the program from *trial* to *fully functional*.

Rent license (aka subscription)

The customer can purchase from you a license that works for only one month. Of course, the customer can also upgrade to a *permanent* license if he wants (and you offer one).

Permanent license

The program can be switched from trial to fully functional with a key received from you, upon purchase.

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 \$10 for each level they want to purchase. Proteus supports up to 16 modules.

Is it difficult to integrate Proteus into an existing program?

What kind of question is this? Of course not!

1. Load the Proteus.DPK file and click "Install".
2. Drag and drop a Proteus component in your application.
3. Set the 'ProductName' to match your product name.
4. Optionally: Set the *ObfuscateRegistry* to True for higher protection or to False for easier debugging.
5. Optionally: Add some code in the *OnSwitchToDemo* event handler to cripple your program (for example disable the Save button).
6. Prepare a Trial certificate using the helper tool provided with the package. This tells the program when to expire (for example after 30 days). Enter the key in the *TProteus.DefaultKey* field in the Object Inspector.
7. At application start-up call *TProteus.Initialize*;

8. Deliver the application as a trial to your customers where it will work for 30 days.

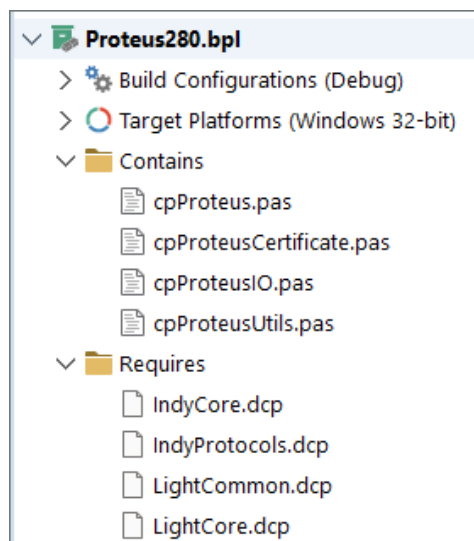
Your application is now a true Trialware application. Ship it to your customers.

Ask a lot of money for it 😊.

Details

Library structure

Proteus is quite a compact library consisting of only four PAS files. However, its basic functionality is supported by the LightSaber library, as seen in the *Requires* section below:



This means that you need to have the LightSaber library installed on your computer, but as we have seen LightSaber is a very lightweight library.

Unfortunately, the Indy library is also required if you want support for MIME encoding (you can replace this with your own encoding if you don't like Indy).

Definitions

Certificates

A certificate is a simple Delphi record that stores 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)

Default certificate

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.

Current certificate

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 `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 (this is typical for applications delivered as Trial). Upon purchase of a license, a key is sent to the customer. When the customer enters that key into the program, a new certificate (let's call it "Full license") appears in the storage area. Proteus will automatically

choose the better (the “Full license”) certificate. The lesser certificate (the “Trial”) will be ignored until the current certificate expires (if ever).

Another example: if the customer was running on a temporary certificate, 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 customer still has time left on his Trial certificate, Proteus will switch to that certificate and run until whatever time was left on that Trai certificate, is used up. If the Trial certificate has no valid time left on it, the program will simply switch to Demo mode, as there are no more valid certificates to use.

Summary

The difference between default and current certificate is that when you deliver a program, it has to have a certificate in it (the expiration data is written into the certificate so Proteus would not know what to do without a certificate).

Later, the customer can receive from you one or more additional certificates. Only one of these can be used by Proteus at a time. This is the current certificate.

Proteus will switch automatically between certificates, choosing the *best* certificate (a certificate that does not expire is always preferred over a *Rent* or *Trial* certificate).

Unlock keys

This is another easy to explain concept: the *unlock key* (or simply “the key”) is a text representation of a certificate.

In other words, the certificate and the unlock key are almost one and the same thing. The small difference is that the *certificate* is a binary Delphi structure (a record) while the *key* is a human readable string, that can be sent to the

customers by email. Therefore, in most cases, the terms *key* and *certificate* can be used interchangeably.

Another way to look at the *key* is as a “recipient” or “transporter” of the binary *certificate*.

Certificate serialization

Proteus converts certificate’s binary data to a human-readable string in a process called *serialization* where it loads certificate’s data into a stream and then passes that stream through a MIME encoder. This is done by the *Serialize* method (see code below).

The opposite of the serialization process (converting a string key back to a certificate) happens in the *DecodeKey* which in the end calls *Deserialize*. The code is pretty much the same, but in we read from stream instead of writing.

```

{ Get certificate data serialized (as string). The CRC is NOT included. }
function RCertificate.serialize: TByteArray;
begin
    var 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;

```

Encryption

Don't worry, the binary data is first encrypted. Also, a checksum is also inserted into the stream. So, the certificate cannot be easily tempered with.

By default, Proteus uses only a basic encryption algorithm but allows you to call your own (better) encryption algorithm if you want. It was designed this way for two reasons:

1. I didn't want to make the Proteus package larger by adding multiple encryption algorithms, because...
2. ...no matter how complex my algorithms would be, there will be some paranoid programmer out there that will cry *"but it is not strong enough for me"* 😊.

Allowing each programmer to use his own algorithm is the best solution!

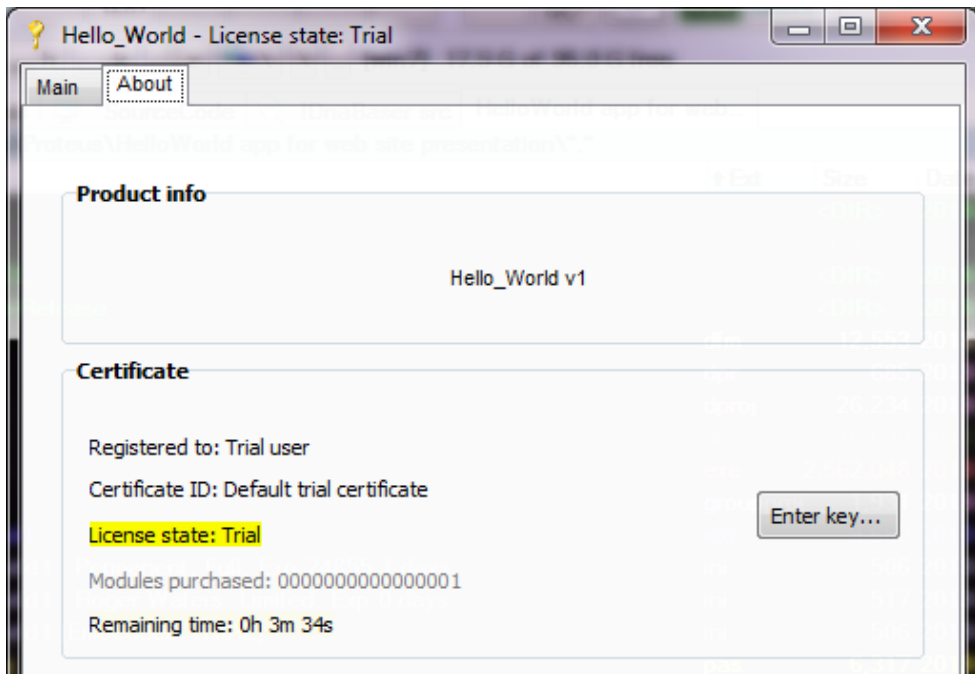
Receiving the key

Upon purchasing a license, the customer receives a key that he must enter into the program. Of course, you don't have to write code for it as Proteus already offer the `TProteus.EnterKey` method which displays the "Enter key" dialog box. But if you really want, you can use your own GUI.

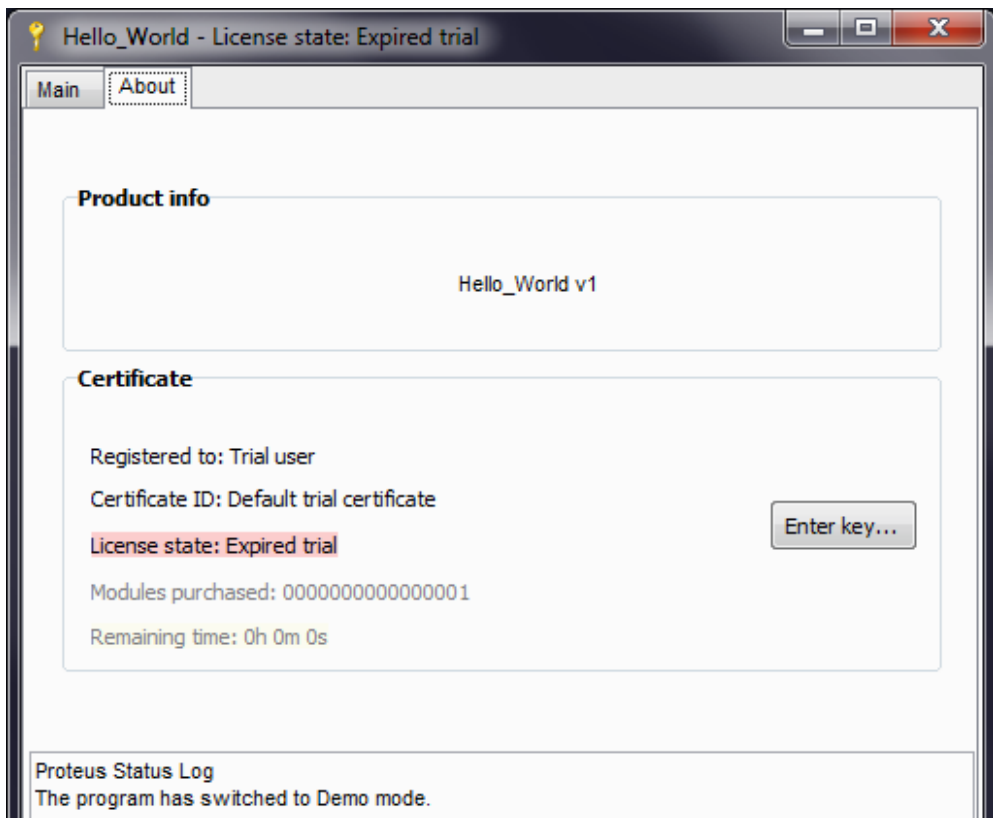
Proteus validates the key entered by the customer. If the key is ok, the program is unlocked.

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 1000 years.



A HelloWorld program in its Trial period.
There are 3min and 34sec left, and counting down.



The program after the trial period expired.

Observe the log.

Hint. If you offer a time-based trial:

- Call *CheckComputerClock* from time to time to make sure the customer didn't set the system clock back. You can call it in a timer or better in a piece of code that is executed regularly such as the *SaveButton* (*before* you allow the customer to save).
- Use the *ShowRemainingTime* function to display how many days are left from the trial period.

Updates of the Trial program

If the customer is in the trial period and he downloads a new trial version (update) from your website and this new trial version has a new trial key, you can decide if Proteus will use the new Trial certificate (get a new *full trial period*) or stick to wherever time was left onto the current certificate. For the latter, set the *UpgradeToNewTrialKey* field to True.

Version-specific keys

You can choose if a key will unlock only the current version of your program or also other versions (older or newer).

For example, the customer has v2.0 of your program and valid certificate (Full, Rent, etc) for it. Then the customer downloads a minor update (v2.1) from your website. Upon installation of the newly downloaded version, Proteus will automatically use the current certificate and the transition to v2.1 be automatic. However, you can dictate how Proteus should behave for major updates, for example v2.1 to v3. Should the customer be able to use his existing v2.1 license

in v3 or not? The answer to this question is stored (by you) in the existing v2 certificate.

In the first case, the transition will be automatic as in the case of v2.1 to v3.

In the latter case, Proteus will refuse to activate v3 and the customer will have to purchase a new license for v3.

Related settings:

UnlockDown: Boolean;

If TRUE, the key works with this version AND all versions below.

UpgradeTrialKey: Boolean;

MAKES SENSE only for 'Trial' certificates. If the customer 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 customer to fall back to a fully functional trial (if there is an existing expired trial certificate, it will be reset and work again every 30days).

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 customer can choose to purchase only one module or both.

You can use *RCertificate.ModulesMask* to define which module to unlock.

Proteus allows up to 16 modules. *cpProteusUtils.pas* offers the tools needed to set/clear a specific bit in the *ModulesMask* word.

```
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 the first and last module use '10000000 00000001'

Storage

Proteus implements a single storage space for certificates: the registry.

However, it is built in such a way that the user can easily expand the library to add support for other storage types such as an encrypted file or even files stored on a remote server.

After we have seen how cool, easy to use, safe, complete, yet flexible Proteus is, let me brag: Can you get a licensing system better than Proteus? 😊

Code structure

It is important to have the code compartmentalized. In Proteus this is done by splitting the code into three main structures: TProteusIO, TProteus and RCertificate.

TProteusIO

The parent class for Proteus is TComponent. This means that Proteus is a visual control that we can drop on a form.

```
TYPE
```

```
  TProteusIO= class(TComponent)
```

TProteusIO class implements basic support for reading/writing the certificate from the current storage space (hence its name I/O).

TProteus

In top of *TProteusIO*, we have the *TProteus* class which does the more refined work of actually handling (decoding) the active certificate and triggering events, for example when the current certificate expires.

RCertificate

RCertificate is a record that holds all the binary data of the certificate. Let's take a look at it.

Note that less relevant fields and methods were removed from the listing below. I have placed the comments under the fields so they can fit better with the way this book is formatted.

```
RCertificate= record
  ID          : String;
  { Unique ID for each certificate type. I use this ID to prevent the customer
    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 customer works. Ex: Company/Academia/Home}

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

```
{ 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 }
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 expiration }
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
customer has tempered with system's clock}

```

```

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

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

{ LICENSE STATE }
function Stollen(StolenKeys: TStringlist): Boolean;
{ Returns true if this certificate is listed in the 'stolen keys' list }
...
end;

```

Support code

The library also offers a bunch of routines to handle the registry, provides predefined messages for the end user, logging, string manipulation, time tracking and bit manipulation. All these have been separated from Proteus code into *cpProteusUtils.pas*.

Self-protection

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 customer tries to reset the Trial period by setting back the computer's clock. An event is triggered in this case. You can write code in the event handler to decide what you want to 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 sent to customers was leaked on serials/cracks website, we can put it in this list, recompile the program and release an update.

Proteus checks the current certificate against the *StolenKeys* at each startup. If the certificate is found in the *StolenKeys* list, Proteus will permanently switch to Demo mode. The customer is not allowed to enter any more keys, even if he tries to enter a valid key. This way we prevent this fraudulent customer from trying to use more stolen keys.

Amnesty key

This special key 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 customer will have to enter a “secret” keyword as a key. You will find the keyword in the source code, in the *ShowEnterKeyBox* function.

What to do if a key was leaked on the Internet?

1. Put the CertificateID and the UserName of the leaked key in 'CheiFurate.txt' using the following format. Be careful not to use spaces around the '=' sign. If you have multiple keys, put one key per line:

```
CertificateID=UserName
```

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

```
Blacklist RCDATA CheiFurate.txt
```

3. Bind the RC file 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.

Certificate ID

The Certificate ID (see *RCertificate.CertificateID* field) is a random text added to each certificate. You can think of it as the thing through which you identify each certificate or as a better alternative for *UserName*.

Unique IDs

In most cases it is recommended to use a unique text for each certificate you generate. One reason for this is the example above where we need to uniquely identify each customer (the *UserName* field is not a good idea since two customers can have the same name).

Another usage for a unique ID is *monthly subscriptions*. In this case we must generate multiple licenses for the same customer (one each month). The key must have a unique ID so Proteus will know which certificate was used up (expired) and which one is new.

Non-unique IDs

However, for special certificates such as the Trial certificate, we want the ID to be non-unique. The Trial certificate is the same for all users, so the ID is always the same. This prevents the customer from getting a new Trial period when he downloads an update (Trial version) from your website.

Obviously, if you *do want* to allow the customers to use the new Trial version, then you need to use a new (unique ID).

Entering the same key twice

Proteus has an internal check, based on *CertificateID*, that prevents the customer from entering the same key twice.

When the customer 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 according to the following rules:

Trial certificates

- When an update is released (for example, v2.0 to v2.1), the CertificateID delivered with this new version should remain the same as the one of the previous versions, to prevent the customer from re-starting the trial period.
- When an upgrade is released (for example, v2 to v3), the CertificateID of the new version could be changed to allow the customer to trial this brand-new version.
- If we want to change the running time of a Trial key (let's say we want to increase it from 14 days to 30 days):
 - If we change the certificate ID - the customer will start a new trial period.
 - If we use the same ID - the customer will use up whatever trial interval he has left from the previous Trial certificate.

Rent (aka 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 above algorithms apply only to certificates that are not time-limited (the *Full* certificates).

Generally, when you generate keys, *append* the current date & time to whatever string you use as CertificateID field, to make sure it is always unique.

Handling multiple certificates

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 according to the current certificate (demo, trial, full).

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

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

In the case of time-limited (Trial or Rent) certificates, when the certificate expires, the program switches to Demo mode, but that certificate still remains the active certificate. It will remain so until a new certificate is entered.

The exception to the above rule happens when the *FallBackToTrial* field is set to True, and the customer uses a Rent certificate. In this case, when the time expires on the Rent certificate, Proteus switches back to the default Trial certificate (the one delivered with the program) and uses whatever time was left on that certificate.

In other words, this allows a customer that rented the program for 30 days to switch back to Trial, once those 30 days passed.

The “Trial expired” message

If the license expired overnight, Proteus would show a "Trial expired" message box during application startup. The program will halt until the customer 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 customer acknowledges that he understands that he is running now in Demo where probably he won't be able to save his work. We don't want to let the customer use the program for a few hours and then not be able to save the work!

OnSwitchToDemo

You can write code in the 'OnSwitchToDemo' event handler to take an action – for example:

- redirect the customer to the Purchase webpage.
- shut down the program
- disable the “Save” menu
- announce to the customer that the trial has expired.

For lazy people, Proteus already offers predefined messages for its events:

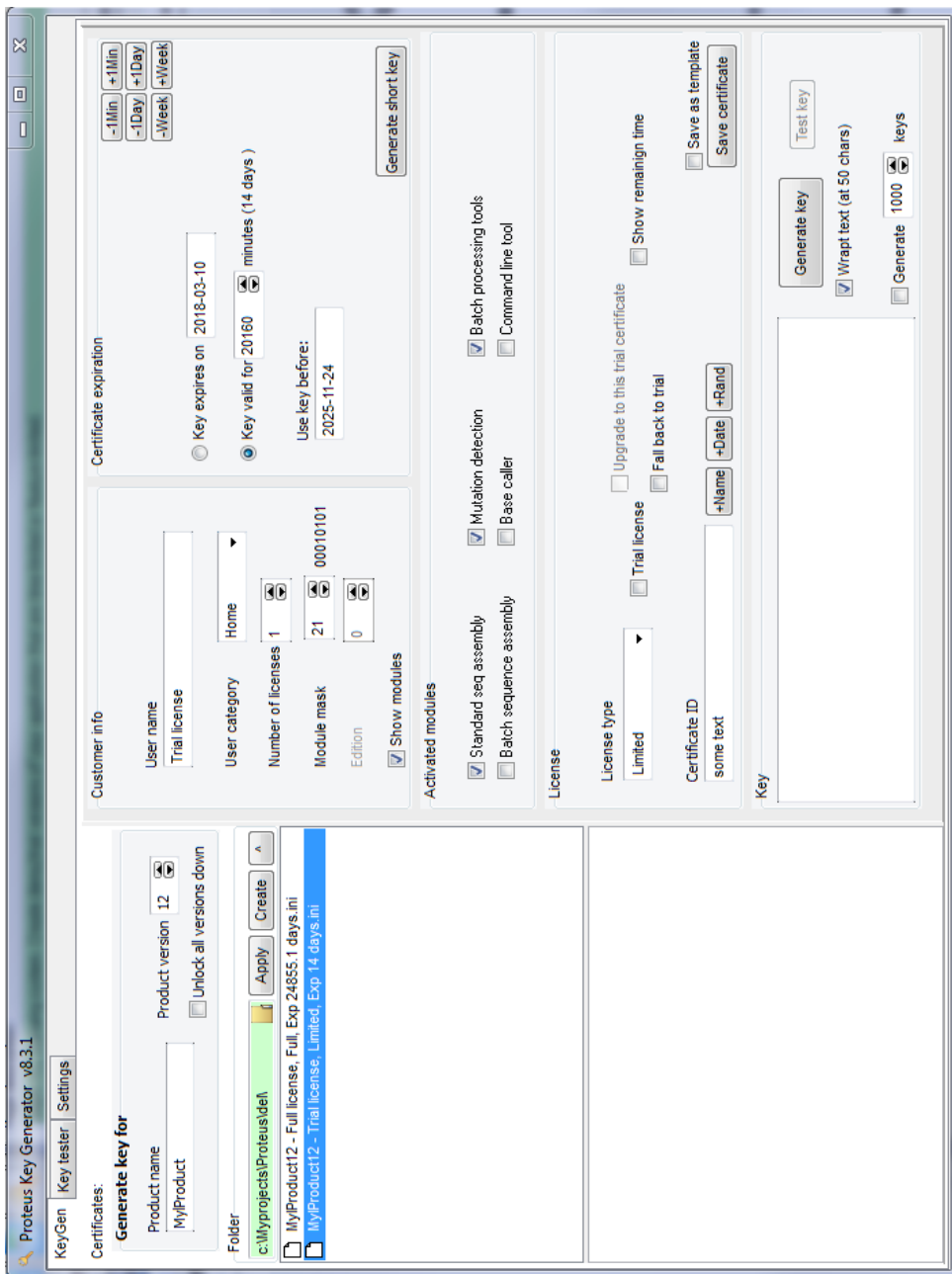
- ShowSwitchToDemo,
- ShowKeyAccepted,
- ShowKeyUnknown,

- SwitchToDemo,
- SwitchToDemoFallBack.

The Key Generator

Below is the screenshot of Key Generator app tailored for my personal needs. As you can see Proteus really offers 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 from scratch. How hard can that be?

Building your own Key Generator

1. Start a new Delphi VCL app.
2. Put a button on it and call the *GenerateKey* 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 *GenerateKey* function above, we initialize a certificate, set *IsTrial* to true and *KeyValidMin* to 60 (minutes). This certificate will now count down from sixty minutes to zero. The string returned by this simple function is a Trial key.

Documentation

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

Online activation server

This is a new feature that allows you can activate/deactivate a license via a PHP activation server.

How does it work?

When a customer purchases a license, you send him an activation key. You also activate that key, on your side, into the PHP activation server.

When the trial app starts it sends a key to the activation server, and the server answers back if the key is active. If the key is active, then your program can switch from trial to fully functional. Each time the program starts it checks this key, and for each check, the server will increment a counter associated with this key.

If the key was used too many times it is an indication that the key was leaked over the Internet and you can inactivate that key.

The Delphi side

The Delphi code is in one single file `cpValidateCertifID.pas` file. For enhanced security it accepts SSL v1 to v3.

It uses Json to exchange to talk the PHP server. It returns the data (customer email, user name, license status, license counter, etc) in a record.

The PHP side

The activation server is in the \PHP folder and consists in a few modules:

- The GUI to add a new license key and its associated customer data (name, email, etc).

- A listener that communicates with the Delphi side.
- A helper tool to enter some demo keys.
- The GUI to list all existing keys and their status.

In the example below we inactivated the license key for user Pepetria because he used the program 325 times in just 3 days (a clear indication that the key was leaked on the Internet):

List of Users

License status for 'testuser2' updated!

Key	Email	License Status	User Data	Toggle License	Counter
UniqueID1	test1@example.com	Active	John Lennon	Deactivate	3
UniqueID2	test2@example.com	Active	Roger Waters	Deactivate	2
UniqueID3	test3@example.com	Inactive	Pepetria	Activate	325

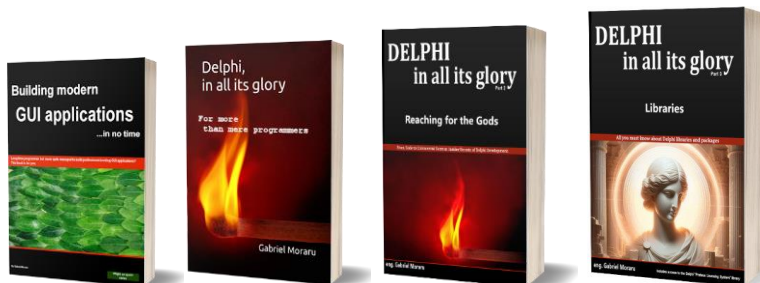
My books

If you liked this book, your next destination could be, of course, one of the other books in the trilogy.

Already published books

- Building cross-platform applications.
- Delphi, in all its glory – Delphi for more than mere mortals.
- Delphi, in all its glory – Reaching for the Gods.
- Delphi, in all its glory – Libraries (this book).

You can find details about my books on www.GabrielMoraru.com



Next books

It took me over three years to prepare, document, write and proof this book. It was a fun ride. If you like it, stand-by for my next books called:

- Your first game (2026)
A book about... well it is obvious about what is this book.
- The things they never told us (2028)

A book about the depths of Delphi. Some kind of continuation of the information missing from the Delphi user manual.

- How to build and sell your own software (2029)
A summary of my experience in starting a company that sells software for molecular biology (DNA sequence assembly).

If you liked this book, please go on Amazon and give it 5 stars.
It will encourage me to speed up the production of the other books.

Contact me!

I spent 6 years writing this trilogy. I hope I did it right. My English is not perfect, but I don't care much about this. What I care about is the correctitude of all information provided in this book.

If you have feedback about this book and **especially** if there are paragraphs that are unclear, please contact me via my website www.GabrielMoraru.com

I promise there will be no email unanswered.