Delphi quick start

Below is a quick-start entry into how to use our license management components in your Delphi applications. Please make sure you read the introduction (https://hexlicense.wordpress.com/) before following this article. You can also have a look at the code (https://hexlicense.wordpress.com/quick-start/delphi-example/) for one of the demonstrations that ship with the package.

The HexLicense package consists of three parts:

- The serial number generator application
- The Delphi license components
- The Delphi plugin menu

The license component package consists of 7 non-visual components. These components can be dropped on a form or a datamodule. They require very little work on your part and can just as easily be dropped into an already existing application as a completely new program.

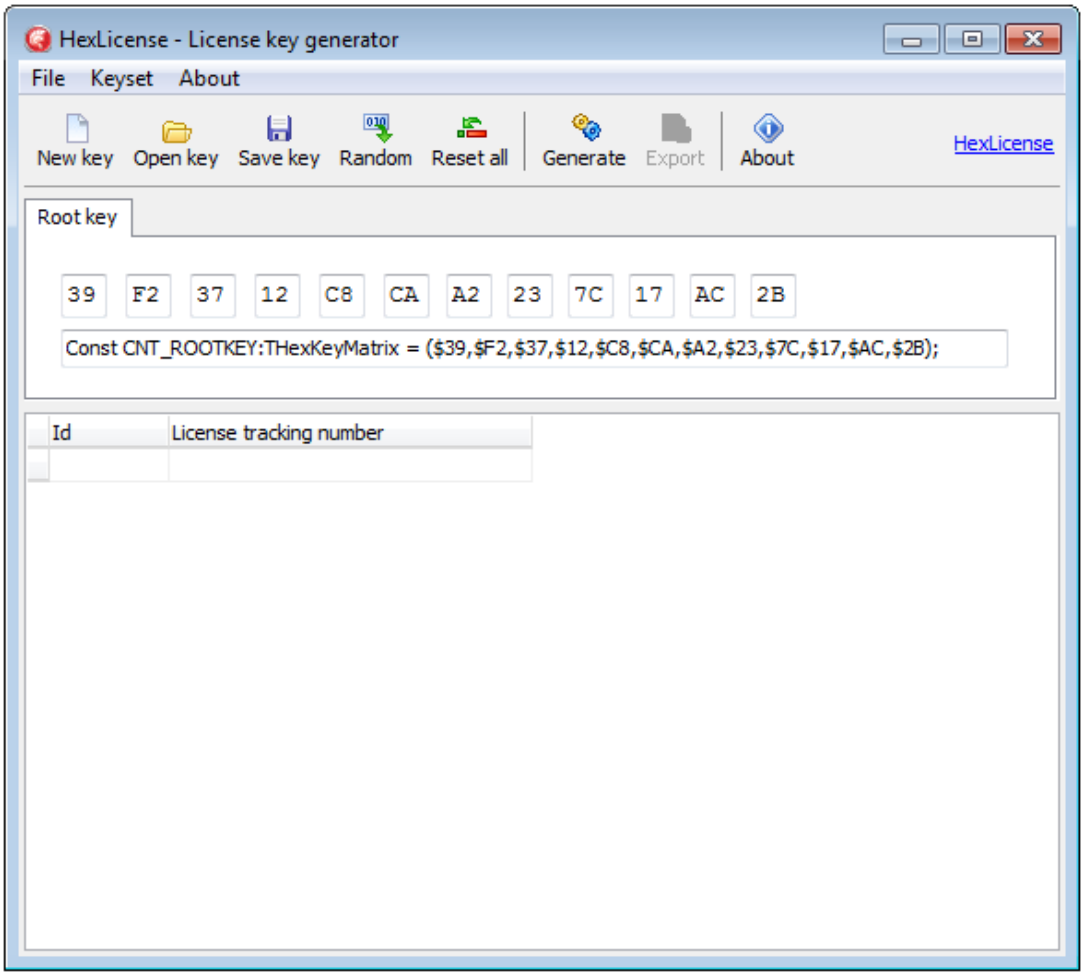
The license components are as follows:

- THexLicense
- THexSerialNumber
- THexSerialMatrix
- THexFileLicenseStorage
- THexRegistryStorage
- THexOwnerStorage

Generating your root key

Adding serial management to your application begins with the serial number generator. The generator (called “keygen” from now on) allows you to generate a random root key, which is the basis from which all compatible serial numbers are derived.

Optionally you may type in your own root-key sequence, but you should try to keep your root-keys as unique as possible.



Using the key creator and license generator is very simple

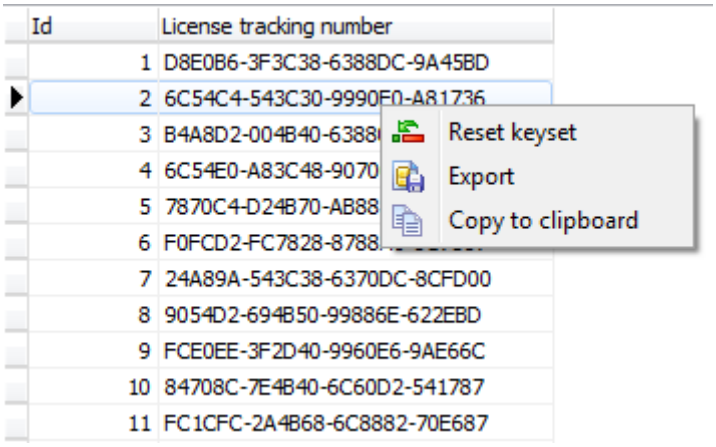
In the picture above the root key is defined by the 12 hexadecimal input boxes. You can use your own values or simply click “new key” to make a random number sequence.

When the key changes the application automatically generates the delphi code you see in the text-edit box below the key values. You need to copy this source-code into your Delphi application and preferably store it in a separate unit. A hexadecimal key is much harder to locate for hackers, since it’s stored with the machine code as opposed to a resource string or text constant.

Generating serial numbers

Once you have generated or typed in a root-key, the keygen application can be used to generate a large amount of unique serial numbers. Each of them derived from this root-key alone, which is also the only numeric sequence capable of validating them.

Note: While you can in theory create a massive amount of serial numbers from the same key, I strongly advice against this. All numbers once large enough begins to suffer from atrophy, meaning that the algorithm must search through a growing spectrum to locate an unused and reproducible combination.



(https://hexlicense.files.wordpress.com/2014/11/keys.png)
Generating serial numbers

It's the exact same problem which make technologies like bit-coin (http://www.bitcoinmining.com/) extremely hard to work with. In that terminology locating a reproducible combination is referred to as "mining".

In short: The more serial numbers you generate from the same root-key, the larger the distance between reproducible number combinations. The distance between matches grows exponentially as you mint and exhaust the base numbers.

As such I recommend that you generate an absolute maximum of 6000 serial numbers per key. But I *strongly urge you* to create a new key for every 1000 licenses you put into circulation.

Preparing your online store

Most online store-fronts support pre-generated lists of serial numbers. Typically you are expected to upload a serial-number file to the server as a normal text-file. Lately some vendors also accept XML and JSON formats as well.


(https://gumroad.com/)
Gumroad is a very good vendor

Personally I would like to recommend Gumroad (https://gumroad.com/) as a vendor. They allow you to upload serial lists and their service is simplicity itself for both customer and provider. Another vendor which accepts serial-number lists is ShareIT (https://secure.shareit.com/shareit/signup.html?sessionid=2722316089&random=e02b8f1740d4091b5c21818eae4f074c), albeit their solution is more troublesome for the customer.

To generate serial numbers, simply click the "generate" button from the toolbar and select the amount. Please note that the more serials you mint, the longer it will take to produce the list (!)

Having generated your serial numbers there are two things you must do:

- Copy the Root-key const
- Export your serial numbers
- Save your key to disk (keep safe)

Root-Key constant

As mentioned, the keygen will automatically create a line of Delphi code for you; A single const array declaration which contains your root-key. This must be copied into the source-code of your application.

When you start your application the THexSerialMatrix component will ask for the root key. It does this through the *OnGetKeyMatrix* event.

Exporting serial numbers

Simply click the Export button from the keygen toolbar and you will be presented with several file-formats to choose from. You can choose to save the serial number list as:

- Text file
- XML dataset (TClientDataset)
- Binary dataset (TClientDataset binary format)
- JSON file

Since the majority of online vendors support plain text-files, this is the best format to pick. I have also added JSON support since I have nodeJS based servers written in Smart Mobile Studio (http://www.smartmobilestudio.com) for HTML5 services.

Save your key to disk

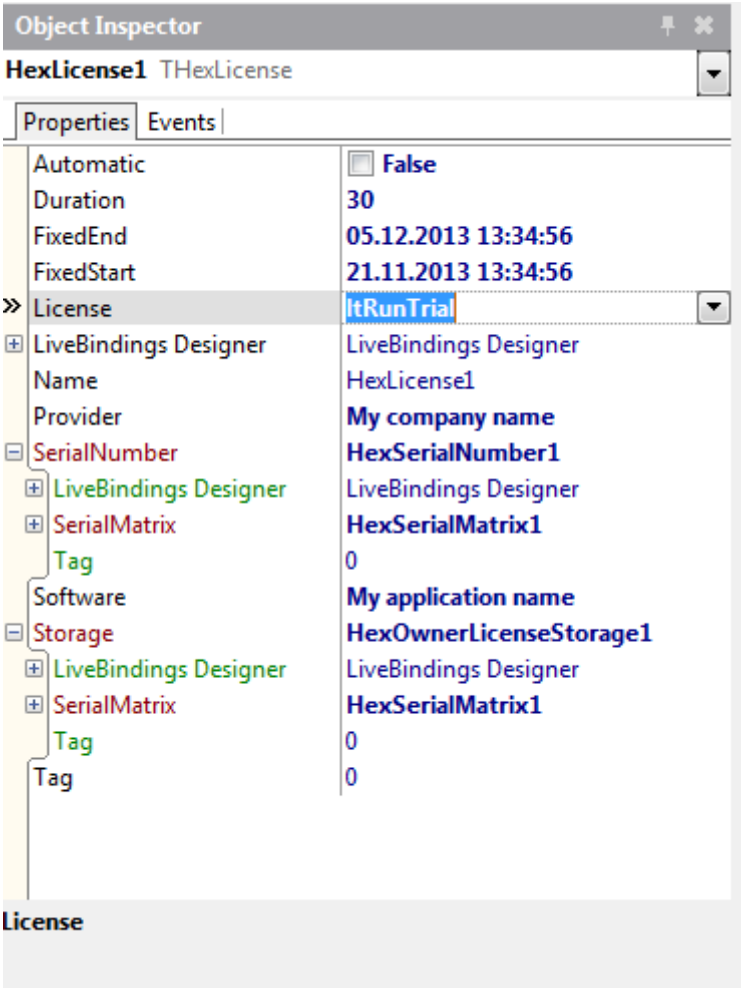
Having created a root-key, generated serial numbers and exported them to the file-system, remember to save your key as well. While you can just type in the root key, saving it in a safe place makes it easier to keep track of your number series.

Please note that saving your key does not save any generated lists. You are meant to use a different root-key ever time you mint new serial numbers.

The Delphi side of things

Now that you have done the preliminary work it's time to work with the components from Delphi. Before we start, a few words about architecture:

Delphi allows your to auto-create datamodules as part of your application's startup sequence. Datamodules are, as you probably know, "invisible forms" (actually they are components with resource persistency) which can host non-visual components exclusively.



(https://hexlicense.files.wordpress.com/2014/11/licprops.png)

Selecting license type

If you are adding license management to an already existing application, chances are that you want to keep it separate from your already existing codebase. If that is the case, simply add a blank datamodule to your project and make sure Delphi automatically creates it when your application starts.

You start by dropping the following components on your datamodule or form:

- THexLicense
- THexSerialNumber
- THexSerialMatrix

Selecting a storage mechanism

Next you need to select a storage mechanism. By default HexLicense ships with 3 different storage adapters, these are:

- THexFileLicenseStorage
- THexRegistryLicenseStorage

- THexOwnerLicenseStorage

I strongly urge you to pick THexOwnerLicenseStorage. Using this storage adapter you can handle the entire storage of license information via ordinary Delphi events. It is imperative that you find a safe place to store the actual license data – here you are only limited by operative system credentials and creativity. Some typical places to store data are:

- Encoded into the pixel-buffer of an image
- Stored as a file inside a zip-file cabinet (password protected)
- Stored as a resource inside an .exe or .dll file (MSDN: UpdateResource API (http://msdn.microsoft.com/en-us/library/windows/desktop/ms648049%28v=vs.85%29.aspx))
- Stored as a TAG chunk of an image file

Connecting the root-key

The component THexSerialMatrix has a single event called "OnGetKeyMatrix". You must respond to this event and return the const array generated by the codegen. An example of such an event handler is:

```
procedure TfrmMain.HexSerialMatrix1GetKeyMatrix
          (Sender: TObject;var Value: THexKeyMatrix);
Const
  CNT_ROOTKEY:THexKeyMatrix = ($4F,$9B,$BD,$7E,
  $79,$8E,$40,$98,$93,$EC,$FB,$D0);
begin
  Value:=CNT_ROOTKEY;
end;
```

This is more or less all the manual coding you have to do. Next you simply need to connect each component to its corresponding published properties.

THexLicense must have it's properties "SerialNumber" and "Storage" connected to the THexSerialNumber and THexStorage component you selected (THexOwnerLicenseStorage is recommended).

THexSerialNumber needs to have the property SerialMatrix connected to the THexSerialMatrix component you dropped on the form or datamodule.

And finally, THexOwnerLicenseStorage must have it's SerialMatrix property connected to the same THexSerialMatrix component as THexSerialNumber.

This may sound very complex but it's very easy and self-explanatory when you examine the properties for these components.

Storage events

The component THexOwnerLicenseStorage exposes only three events. These are:

- OnDataExists

- OnReadData
- OnWriteData

The component does not really care how or where you store the license data (which is just a small chunk of encrypted data), it only cares that these three events are handled properly.

When your application starts the first event to fire on this component is OnDataExists.

If no data can be found (meaning, that it's the first time the user starts you program) a blank license is initialized and immediately stored though OnWriteData.

What is written inside the license file greatly depends on what type of license you want to use. As of writing you can choose between:

- ltDayTrial
- ltFixed
- ltRunTrial

ltDayTrial

Day trial allows the user to test your product for a finite amount of days. Typically 14 or 30 days of free use, before the customer must buy a full license if he or she wants to continue using your application. The amount of days is defined by the property "duration" of THexLicense. The program starts to count down the first time the application is executed. It also checks if the clock has been held back.

ltFixed

While rare, this type of license allows you to compile a fixed start and stop date into your application – and your program will only work within the range of those dates. The start and stop-dates are defined by the "fixedStart" and "fixedEnd" TDateTime properties exposed by THexLicense.

ltRunTrial

This kind of license allows the user to start your application a finite amount of times, typically 100 times for commercial applications or 200 times for share-ware. The amount of runs is defined by the "duration" property of THexLicense.

You define your license type completely through the properties of THexLicense. This component also exposes all the events you need to handle activation, trial "time out" and other measurements. By setting the property "Automatic" to true the license session is initialized immediately when the application starts. If you want a finer grain of control over how the license management system works, you can set disable to false and activate the component by calling the method "BeginSession()" whenever you mean it's most appropriate for your program.

Final words

HexLicense ships with two demonstration programs that show exactly how to deal with everything explained so far. Each demonstration is heavily documented (in code) to make the process as straight-forward as possible; even for novice Delphi developers.

It is important to recognize that HexLicense was designed to deal with serial number generation, validation and recognition. It was made to keep ordinary customers from abusing your licenses – and also for you to have a uniform way of linking customers to their licenses in order to protect their investment.

But there is no such thing as an un-crackable product. If a company makes such a claim I would avoid them at all cost, because lying about something so fundamentally incorrect is simply fraud.

> *"Bottom line: only with a secure hardware dongle, implemented correctly, can you guarantee against cracking" -Source: John Browne, Wibu systems (http://www.wibu.com/us)*

Microsoft spent millions trying to come up with a scheme to make Windows "un breakable", yet only hours after Windows 7 was released – a fully working cracked pirate version could be downloaded from the internet. Same with Windows 10 and all previous version of their operative system.

The same goes for huge corporations like Adobe, Apple and all the others. The open-source tools for disassembling and reverse-engineer software is just as evolved as the products to compile and build software.

If you are going to put your trust in a company or security company, at least make sure they are honest and genuinely interested in buying you time, which is what serial number protection is all about.

In short, serial number protection gives you the following advantages:

- Non technical users are stopped from abusing their license
- Serious customers recognize the value of a proper license
- By carefully planning your release cycle and the use of keys, you can minimize loss even if your product is hacked. Only the hacked build will be vulnerable, which represents only a single key combination

HexLicense makes it harder and more time consuming for hackers to break your software, they have to go out of their way to do so. This time, the point of sales window, is basically where you make your earnings. The larger the windows of opportunity, the more sales you can secure.

Steps you can take

There are several steps you can take to make your product more time consuming to break, like refusing to run the application if a system-wide debugger is running (http://msdn.microsoft.com/en-us/library/windows/desktop/ms680345%28v=vs.85%29.aspx) and also to disable memory dumps. These steps are publicly known, and while effective – they wont stop the most experienced hackers from getting at your code. **Nothing will. Thats just reality.**

But at least with HexLicense you wont make it easy for them, and it will stop ordinary users from copying your work without any resistance.