**.net Licensing Manual**

|  |  |  |
| --- | --- | --- |
| author | date | publish |
| Eugeigne | 2021/10/15 | Draft |
| Eugeigne | 2021/10/19 | Draft |

# Device Information

Licensing is initiated from getting unique machine information.

C# standard library supports functions to get device information of the local computer.

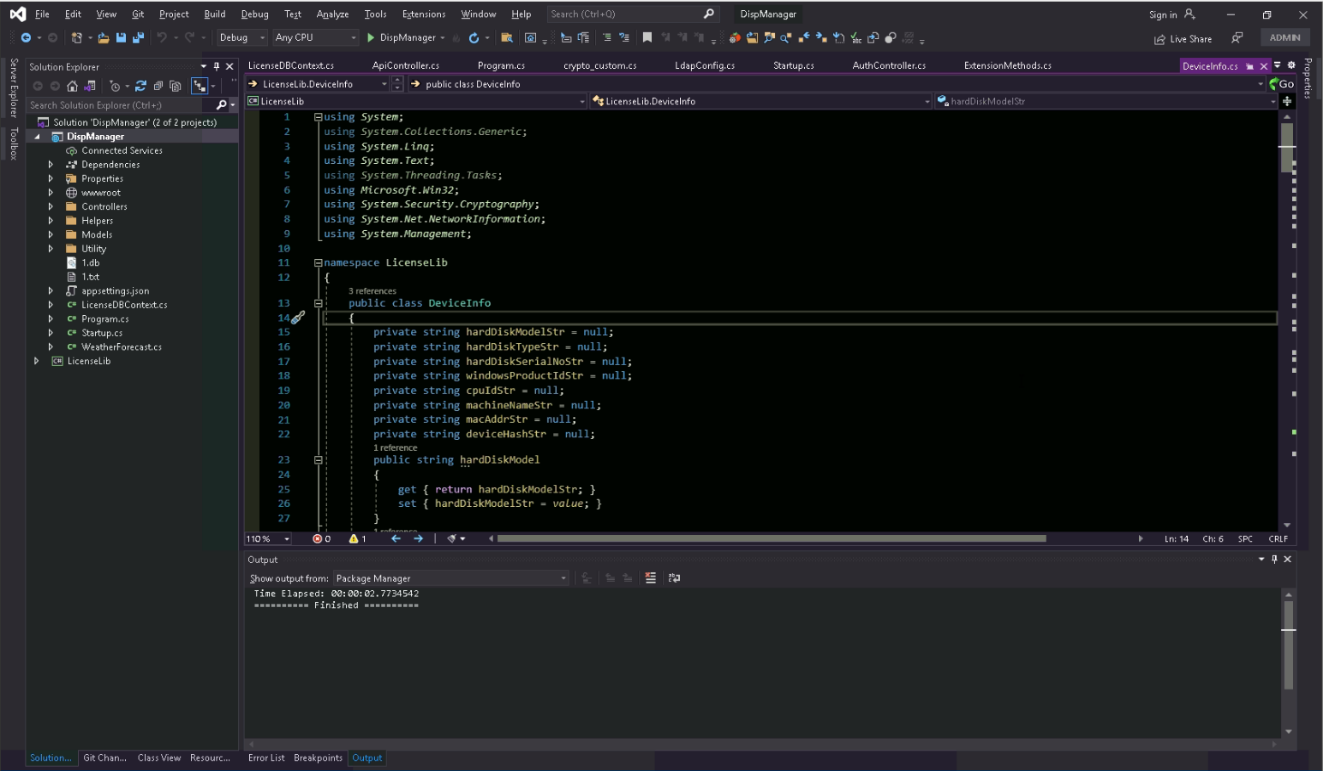
Device information is composed of the following information.

|  |
| --- |
| Hard Disk Serial Number |
| Windows Product Id |
| CPU id |
| Computer Name |
| MAC address |

Computer Name could be changed by a user and MAC address could be updated if the NIC is changed.

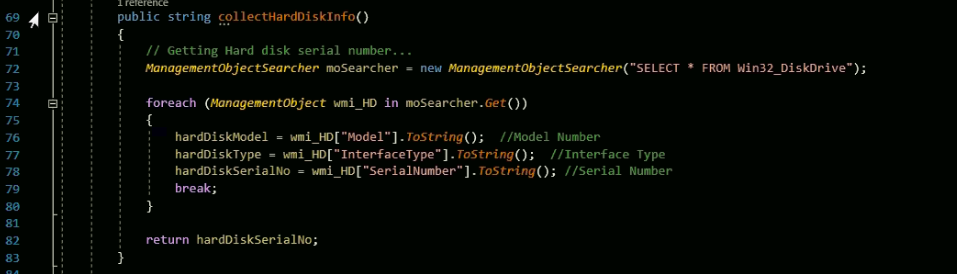
So we barred “Computer Name”, “MAC address” from composition of Device information.

Source file that gets device information is “DeviceInfo.cs”.



## Hard disk serial number

We can get hard disk serial number from ManagementObjectSearcher. We implemented it as “collectHardDiskInfo()” function



Here variable “hardDiskSerialNo” is actually getter/setter but is regarded as a variable. This is the string that represents hard disk serial number.

Ex. “ WJB08LPZ”

## Windows Product Id

Windows Product Id is read-only under the provision of installation.

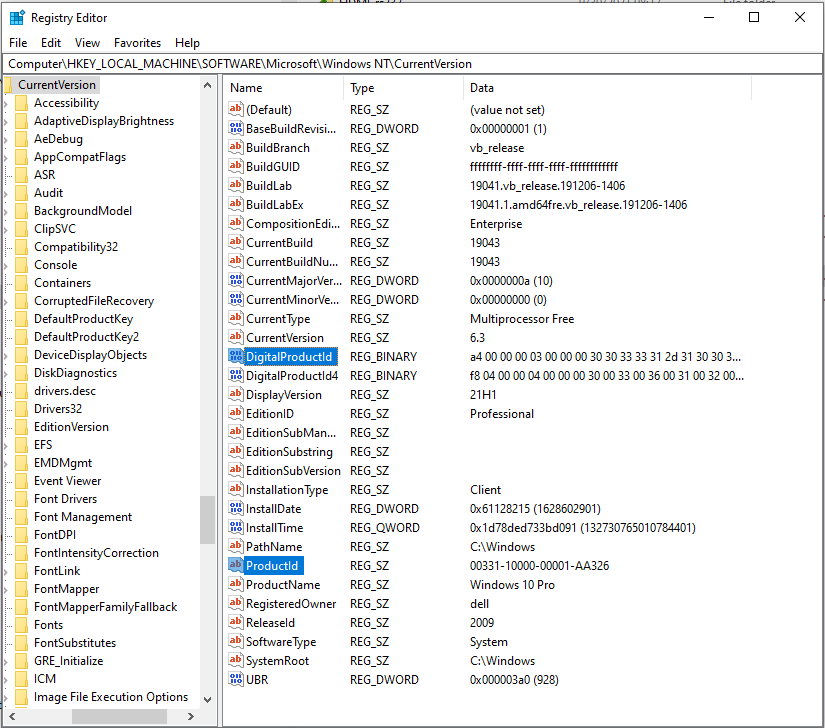
It can be retrieved from the registry key.

“HKEY\_LOCAL\_MACHINE/SOFTWARE/Microsoft/Windows NT/CurrentVersion”

This registry key has 2 values for windows product id.

“DigitalProductId”: It is a stored hex value of windows product id.

“ProductId”: It is a stored string value of windows product id.

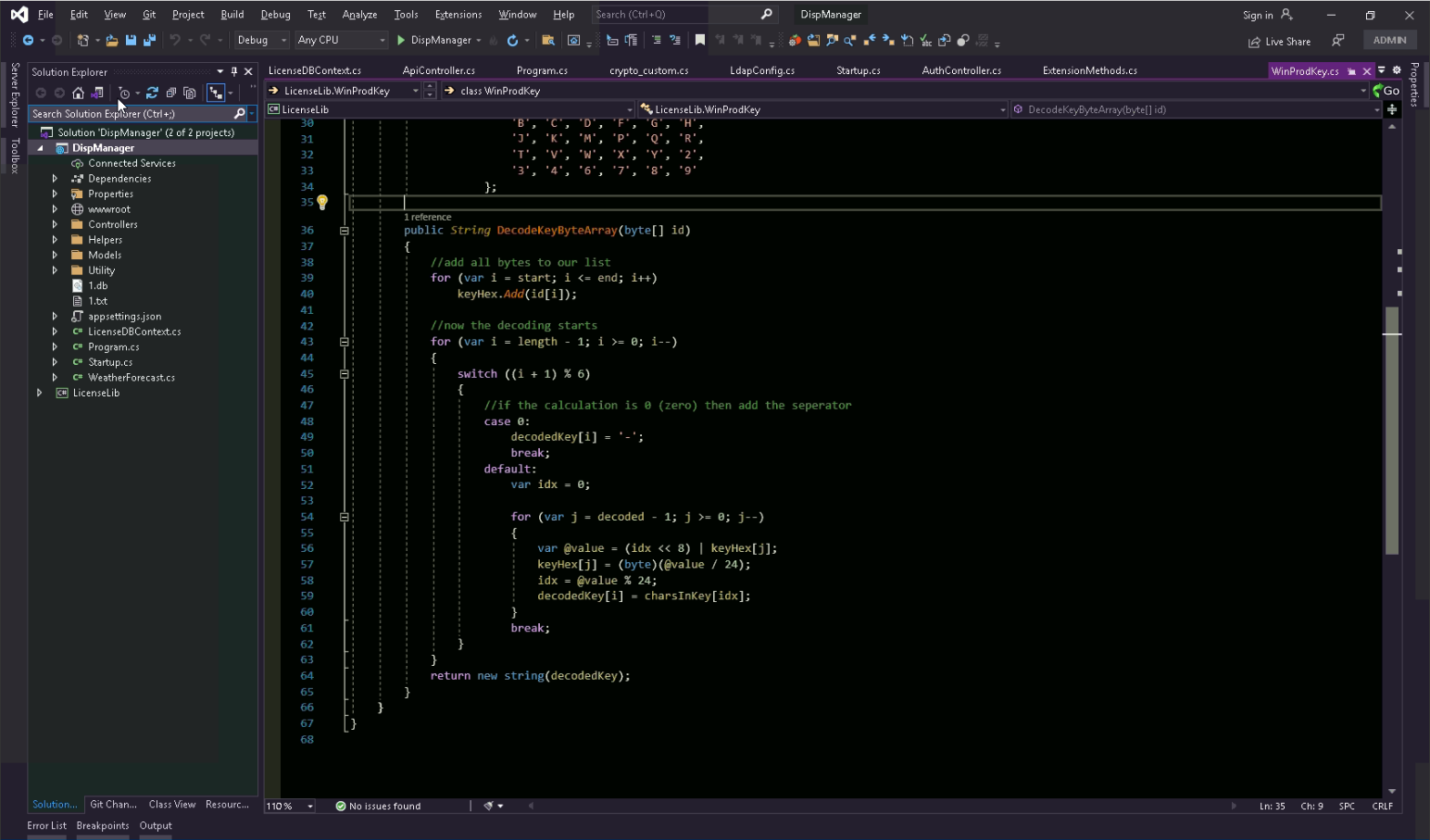


We can use either of both items for windows product id so that we used “DigitalProductId”.

It is hex-format string so that it should be converted to string format.

Conversion is done by “WinProdKey” class.

This class has a member function “DecodeKeyByteArray()” that converts hex-format string to string format.



“DeviceInfo” class has a member function to retrieve “Windows Product Id”.



Here variable “windowProductId” is actually getter/setter but is regarded as a variable. This is the string that represents windows product id.

## CPU id

CPU id is a string representing the unique CPU identifier.

“DeviceInfo” class has a member function to retrieve the CPU id.



Here variable “cpuId” is actually getter/setter but is regarded as a variable. This is the string that represents CPU id.

## Computer Name

Computer Name could be changed by a user so that we don’t use it for device information(See section 6)). However, we describe how to get Computer Name. It’s simple. “DeviceInfo” class has a member function to retrieve Computer Name as a string.



Here variable “machineName” is actually getter/setter but is regarded as a variable. This is the string that represents Computer Name.

## MAC address

MAC address is a physical address of NIC. A computer might have several NICs or a user can substitute it by another one so that it could be changed anytime. So we barred it(see section 6)) from getting device information but we now describe how to get it.

“DeviceInfo” class has a member function to get it.



Here variable “macAddr” is actually getter/setter but is regarded as a variable. This is the string that represents MAC address of the first NIC installed on the computer motherboard.

## Device Information String

We can now get hardware information for the local computer. Now we might get device information by calculating hash string of all hardware information.

At first, we collect all strings from hardware information. As mentioned above, Computer Name and MAC address are not used for device id string.

#### How to bar hardware information from composition of device id string:

Now as you can see at the picture below, computer name and mac address are barred from composition of the basic string for device id string.

Function getHashStringFromInfo() in DeviceInfo.cs of LicenseLib is the one of getting device id string.

The first parameter “strHdSn” is the hard disk serial number string.

The second parameter “strWinProdId” is the windows product id string.

The third parameter “strCPUId” is the CPU id string.

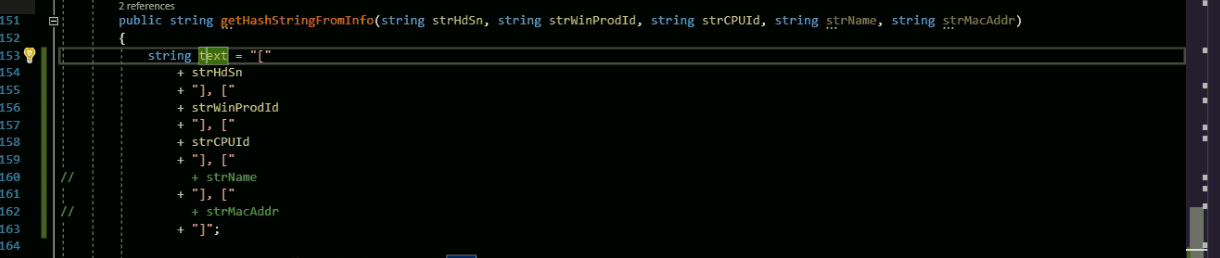
The fourth parameter “strName” is the Computer Name.

The fifth parameter “strMacAddr” is the MAC address of the first NIC installed on the computer.

If you want to except “hard disk serial number”, then please comment or remove the line 154. The parameter “strHdSn” is excepted from making “text”.

Then “hard disk serial number” is barred from composing of device id string.

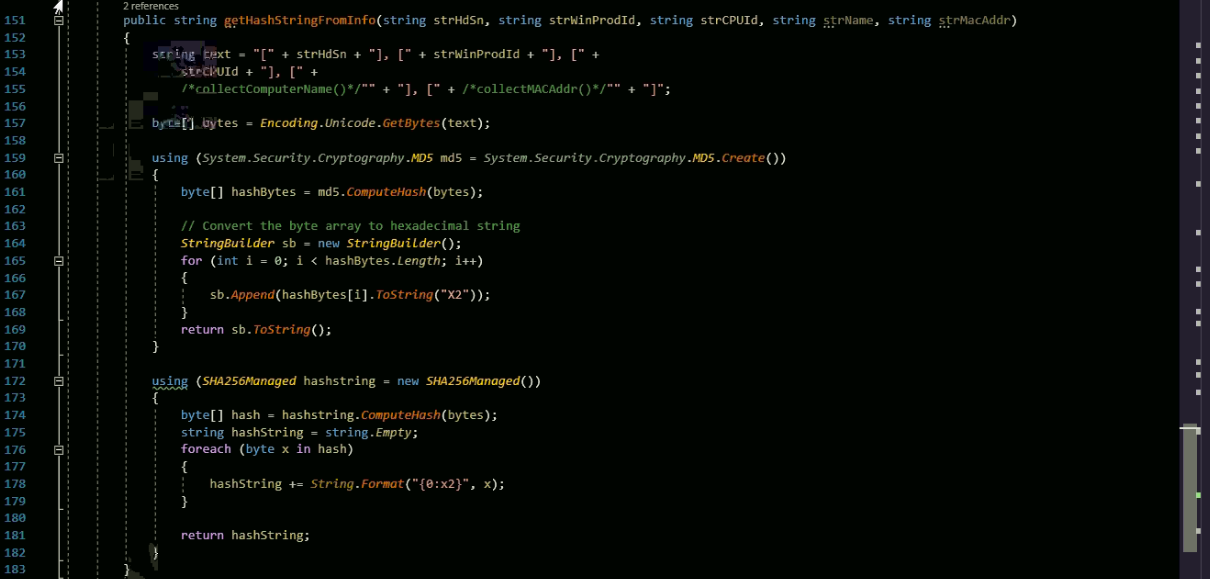
Windows Product Id is at the line 156 as well, CPU id is at the line 158, Computer Name is at the line 160, MAC address is at the line 162.



Basic string for device id string

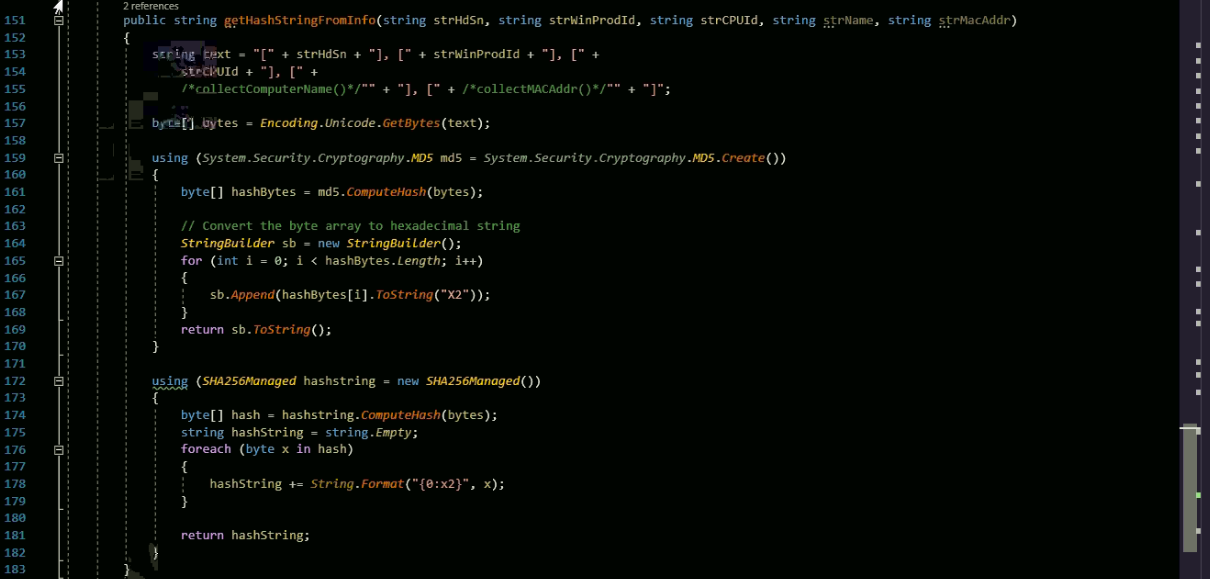
After getting basic string for device id string, we can make device id string from the variable “text”.(See the line 153 at the picture above)

We might use MD5 or SHA256 in this function.

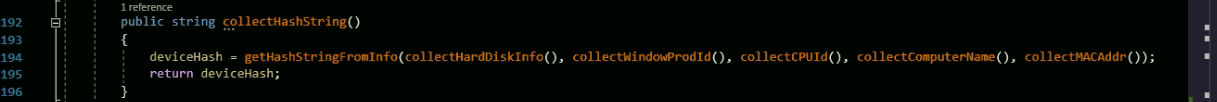


We calculate hash string by MD5 library.

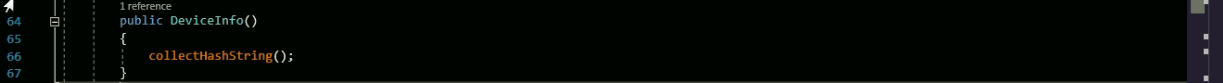
If you want to use hash algorithm SHA256, then please remove the code for MD5 as below.



We finally store device information in the variable “deviceHash”.



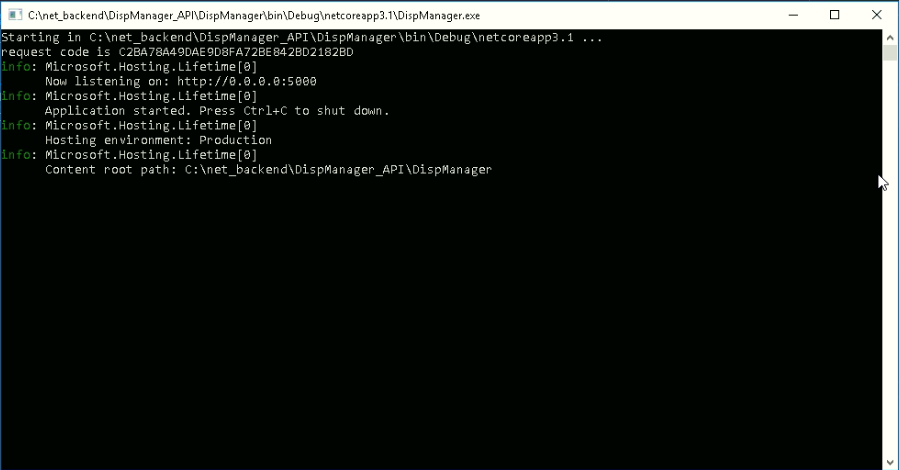
“DeviceInfo” class constructor calls “getHashStringFromInfo()” function and stores the string in the variable.



Variable “deviceHash” is actually getter/setter but is regarded as a variable. This is the string that represents device information of the local computer.

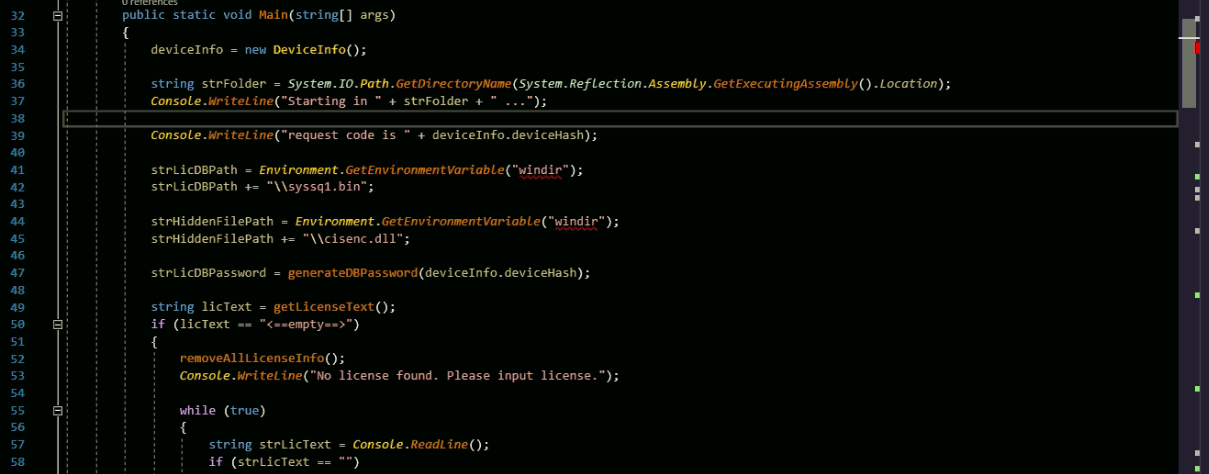
The variable of the DeviceInfo class stores the device id string for the machine.

If you run .net app, you can see the request code like this.



This is the device information we got.

We coded it in .net project as



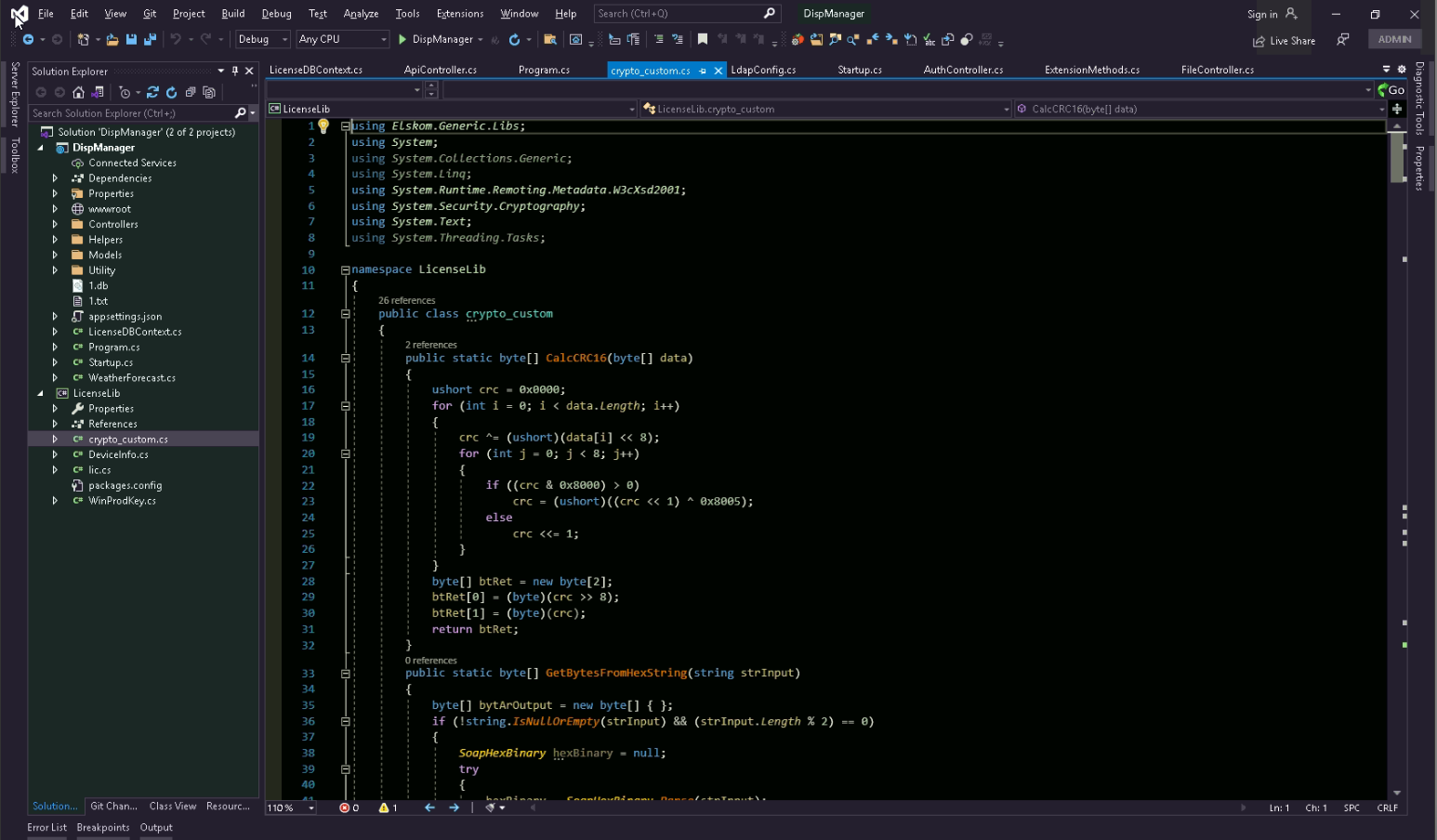
Displays device id string on console window

Gets Device id string

# Licensing algorithm

Licensing algorithm is similar to Cpp project.

Basic functions are defined in “crypto\_custom” class of LicenseLib project.

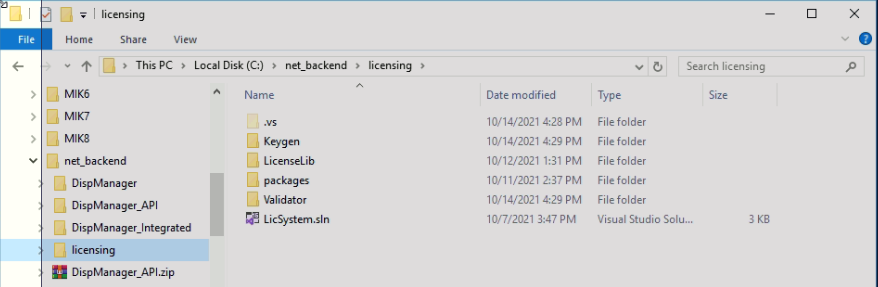


Basic functions are CRC16, SHA256, RSA2048, Blowfish, Base64.

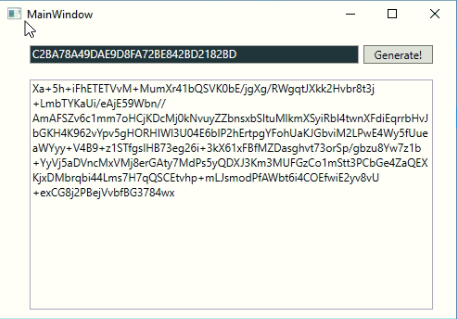
This library is used for both of generation of activation code and validation of activation code, i.e. it is used in keygen.exe and .net app.

## Generation of activation code

Generation of the activation code is done in keygen.exe. This application is built from the project “Keygen”.

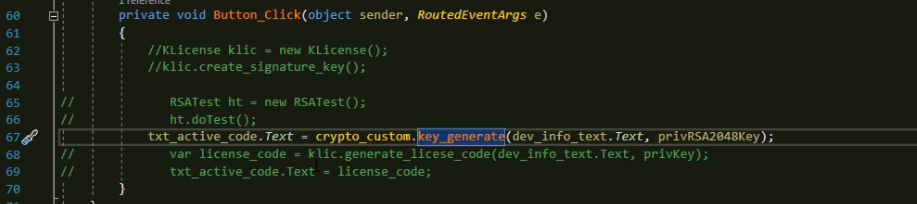


If you open the solution “LicSystem.sln”, you will see the project “Keygen”, “LicenseLib” and “Validator”. “Keygen” project makes keygen.exe to generate an activation key. The picture below shows the running result of keygen.exe

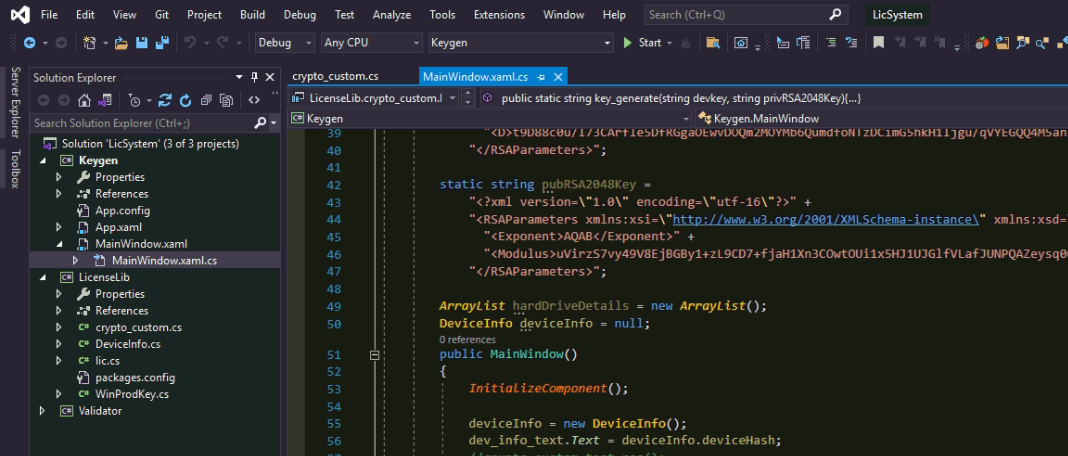


The project “Keygen” is very simple project which makes activation code based on the project “LicenseLib”.

If a provider clicks the button “Generate!”, then Button\_Click() is called as below.



Here calls the key\_generate() function of the class “crypto\_custom”. This class is the one of “LicenseLib” project.

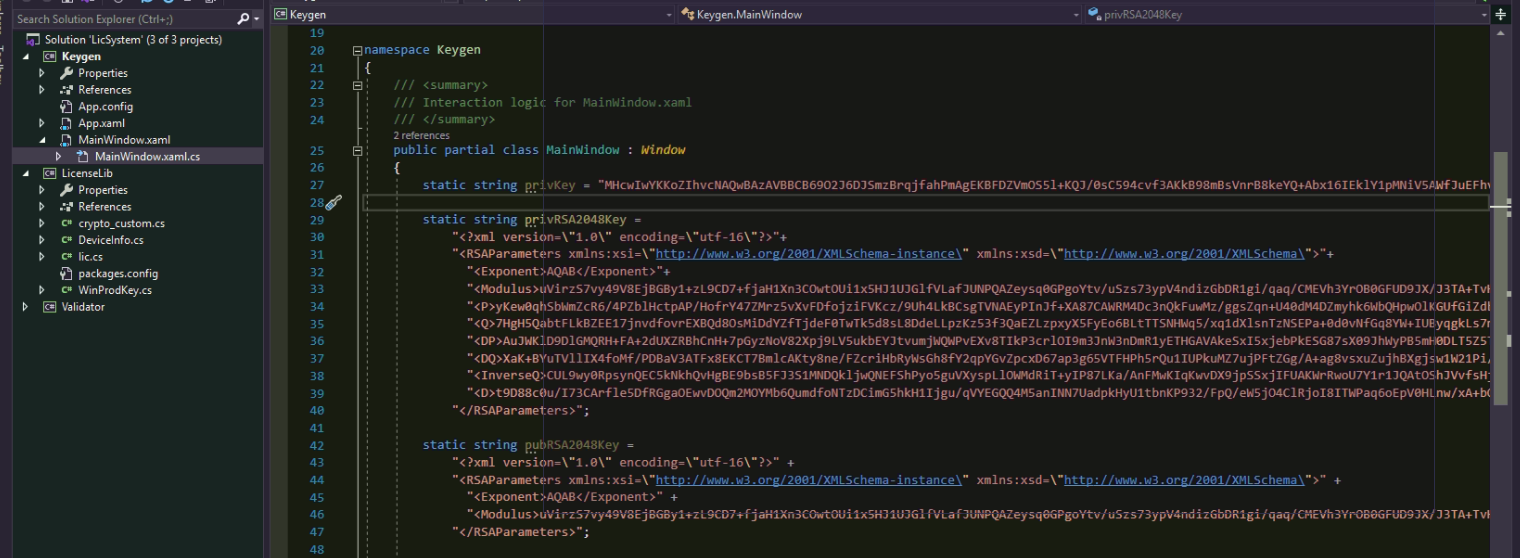


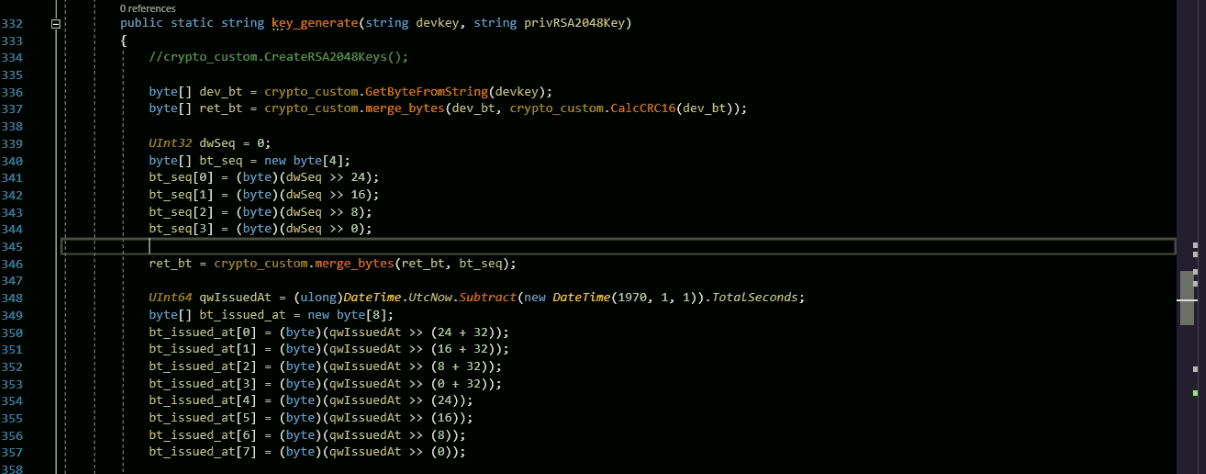
In crypto\_custom.cs file, there is a class “crypto\_custom” which manages encode/decode actions.

This class has the function to generate activation code and check activation code as well.

To generate activation code, are needed request code(device information) and RSA2048 private key.

This private key is the one that only a provider can have in keygen.exe. Project “Keygen” will use “LicenseLib”, and “Keygen” has private key in its project code.





### key\_generate()

Generation of activation code is done by 3 steps.

|  |
| --- |
| Construction of information bytes |
| Generating RSA2048-signature |
| Blowfish and base64 encoding |

* Composition of information bytes

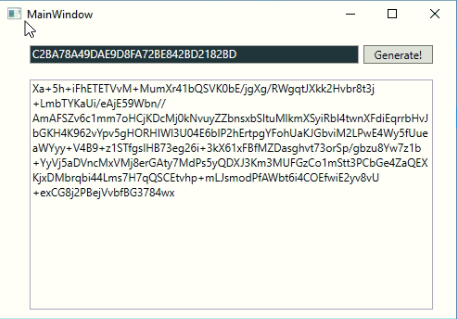
It is composed of the following data sequence.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Dev info | CRC16 | sequence | Issued at | period | flag | padding |
| 32(bytes) | 2 | 4 | 8 | 4 | 4 | 2 |

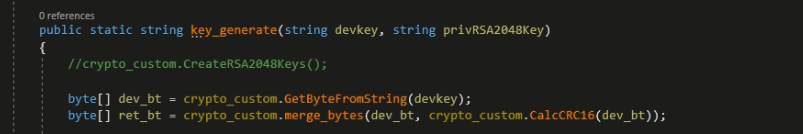
Total length of information bytes is 56.

The length should be multiple of 4 because blowfish algorithm requires it.

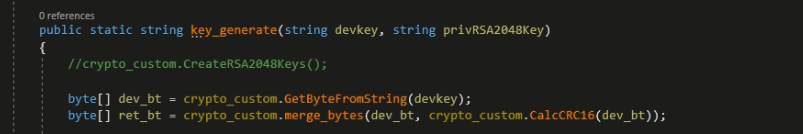
Device information is described in previous section. Provider inputs device id string in the text box of Keygen.exe



It is MD5 string so that 32 bytes.



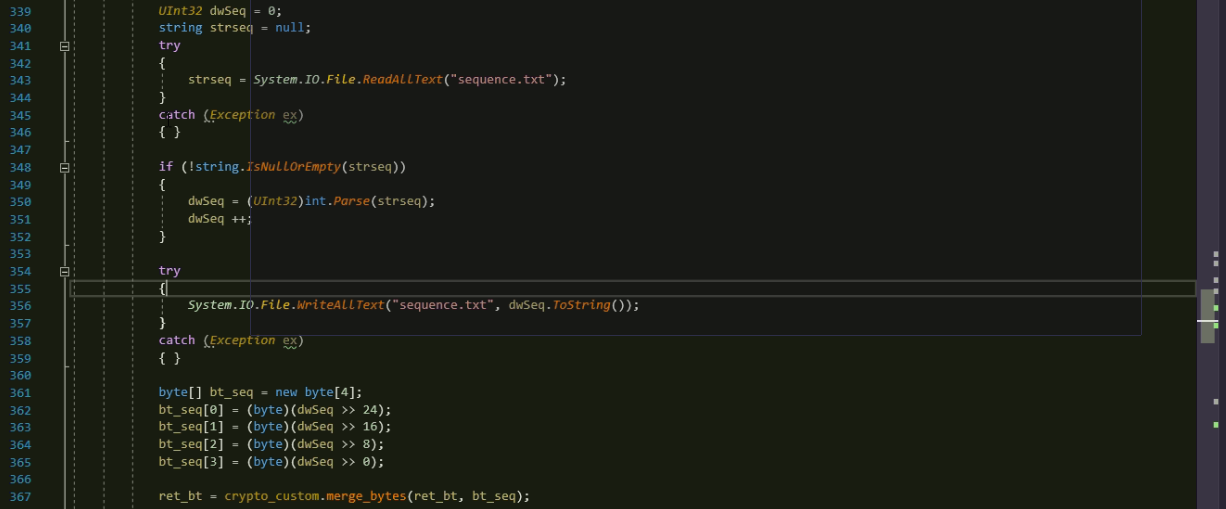
CRC16 is calculated for device information and it is appended to information bytes.



Sequence is the incrementing 4-byte number of the activation code. Whenever a provider makes activation code, “sequence” is incremented by one. For example, “sequence” 45 means that a provider made activation codes 44 times by the keygen.exe, and the current activation number to make is 45th one. Now we set it always to be 0.

The last sequence number is saved in the file “sequence.txt” in the same directory as the keygen.exe file.

Whenever you make activation code, keygen.exe read last sequence number from “sequence.txt” and increment by one, saves it back in “sequence.txt” file.



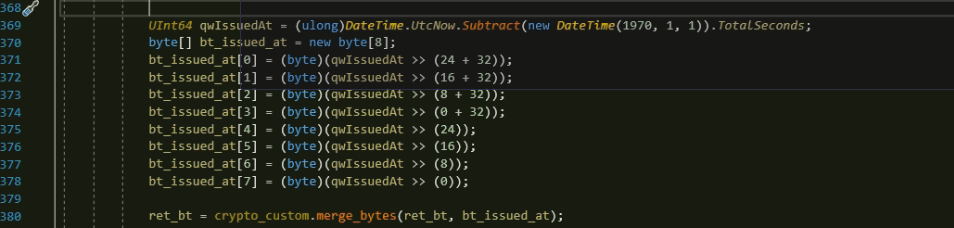
Append sequence number to byte array

Save back in “sequence.txt”

Increment by one

Load last sequence number from “sequence.txt”

“Issued at” is total seconds elapsed since 1970/1/1 00:00:00. We calculate the total seconds between 1970/1/1 00:00:00 and now to append to byte array. It is 8-byte number.



Appending to byte array

Total seconds since 1970/1/1

“period” is the expiration period since “Issued at” timestamp. It is 4-byte number which is total seconds of the expiration period. In the following code, period is 30 days.

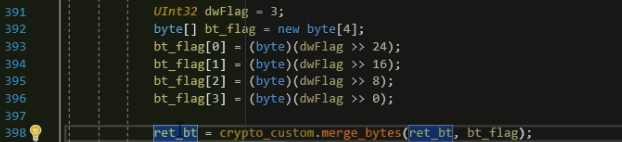


Period in seconds

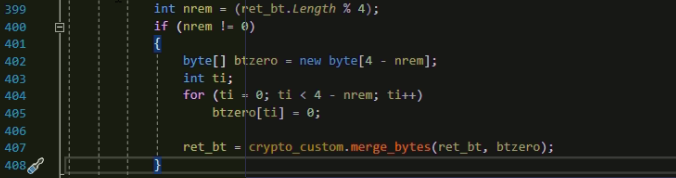
If you want to change the period as 1 year, then please change the code as

|  |
| --- |
| dwPeriod = 365 \* 24 \* 3600; |

“flag” is the 4-byte number that represents the additional information of the license. Here we use 3.



And here is the code for 4-byte-aligning byte size. To encrypt by blowfish, the size of data should be multiple of 4. So if the size of byte array is not multiple of 4, we add zeroes to be multiple of 4.

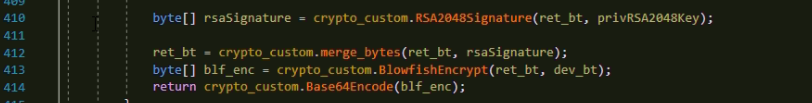


We calculate RSA2048 signature of the byte array.



We append this signature to the information byte array. The length of the RSA2048 signature is 256 and total length of information bytes is 56 so that the byte array will be 312 bytes.

We do blowfish-encrypt for this byte array with the encryption key of device id string bytes.



Device id string bytes

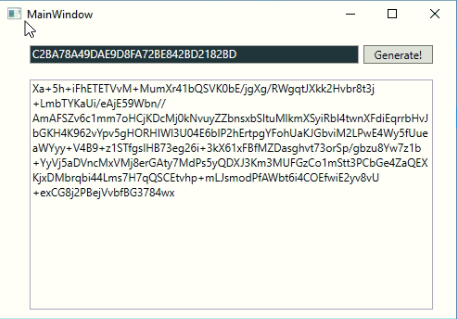
Total byte array

Finally we get activation code string by encoding from Base64.

Total steps are as follows.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Device id string | | | | | | | |
| Device id byte array(32 bytes) | | | | | | | |
| Dev info | CRC16 | sequence | Issued at | period | flag | padding | RSA2048 signature |
| 32(bytes) | 2 | 4 | 8 | 4 | 4 | 2 | 256 |
| Blowfish encrypt(312 bytes: key => Dev info) | | | | | | | |
| Base64 encode | | | | | | | |

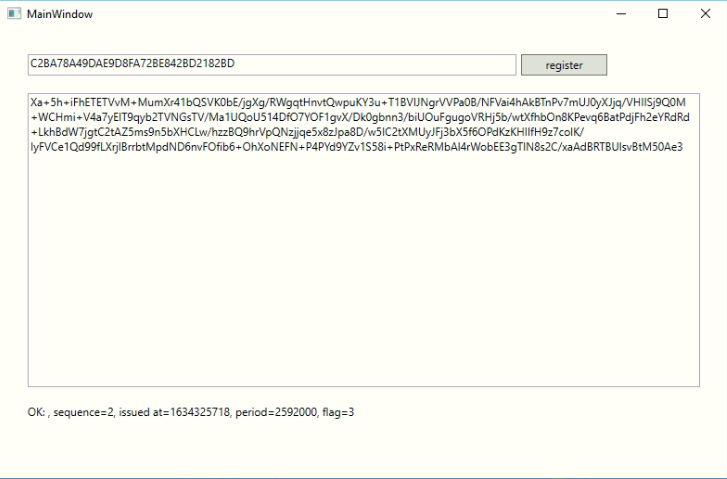
The function key\_generate() returns activation code string and it is shown in textbox of keygen.exe.



## Validation tool of activation code

In the solution “LicSystem.sln”, there is a project “Validator”. This project is to check the validity of the activation code for request code(device id string).

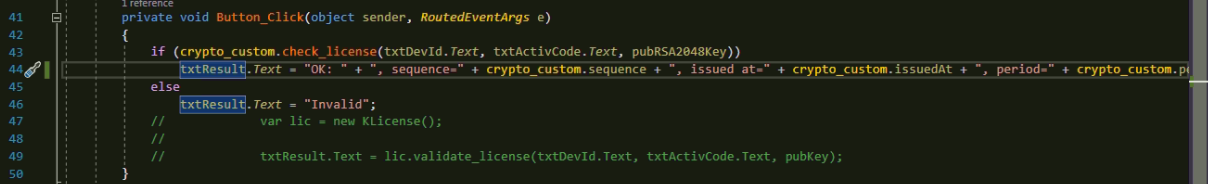
“Validator.exe” is the application of the project “Validator”.



This project is simple one that uses crypto\_custom class of “LicenseLib”.

If user click “register”, then it checks validity of the activation code for the request code and shows the result.

The code for button click is as below.



The result is “OK”, or “Invalid”. If the result is “OK”, then it show additional information of the activation code.

For example, please look at the picture above.

The result is “OK: sequence=2, issued at=1634325718, period=2592000, flag=3”.

This means the activation code was made secondly(sequence=2), and published 1634325718 seconds since 1970/1/1, it will expire at 1634325718 + 2592000 seconds since 1970/1/1, flag is 3.

In this function, check\_license() of the class “crypto\_custom” is used. This returns “true” or “false” which means the activation code is valid or not.

This function works in reverse order of key\_generate() function.

### check\_license()

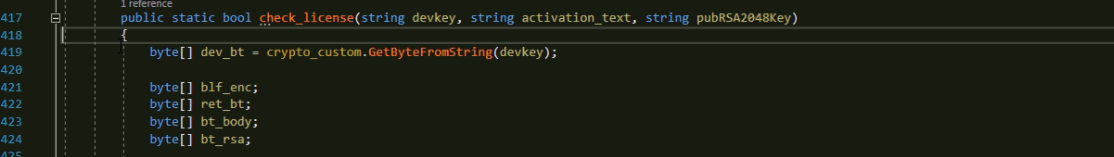
check\_license() has three parameters, device id string, activation code, RSA2048 public key.

RSA2048 public key is stored in “Validator” project.

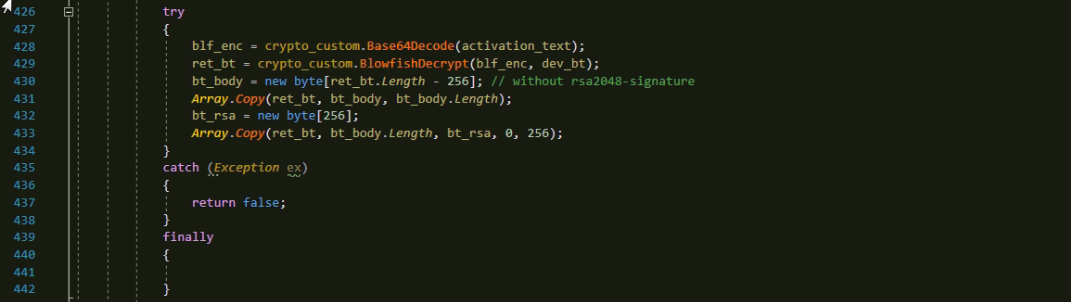


This public key should be paired to the private key in “Keygen” project.

At first, we get byte array from device id string.



And decode by base64, decrypt by blowfish with key of device id byte array, split it into body and RSA2048 signature part.



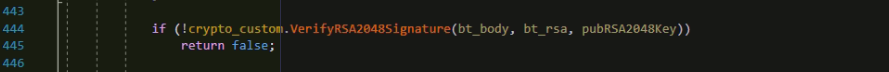
Split second part: RSA2048 signature

Split first part: information bytes

Blowfish decrypt by key of “dev\_bt”

Base64 decode

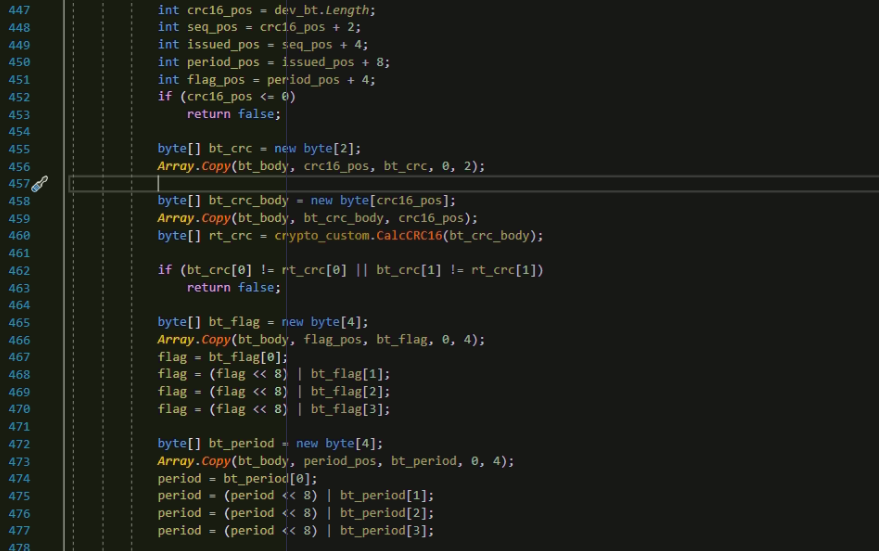
We check validity of RSA2048 signature.



* Decomposition of information bytes

And we split CRC16, sequence, issued at, period, flag parts by the following the structure of the information bytes.

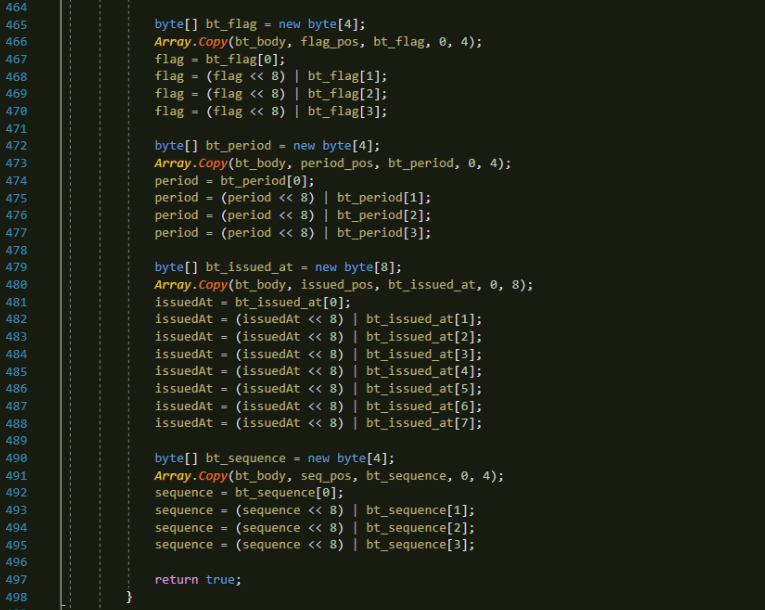
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Dev info | CRC16 | sequence | Issued at | period | flag | padding |
| 32(bytes) | 2 | 4 | 8 | 4 | 4 | 2 |



Compare CRC16 checksum

Split CRC16-payload part: 32 bytes

Split CRC16 part: 2 bytes



Get “sequence”

Get “Issued at”

Get “period”

Get “flag”

After splitting into several variables, function succeeds. All decrypted parameters are static ones in crypto\_custom class.

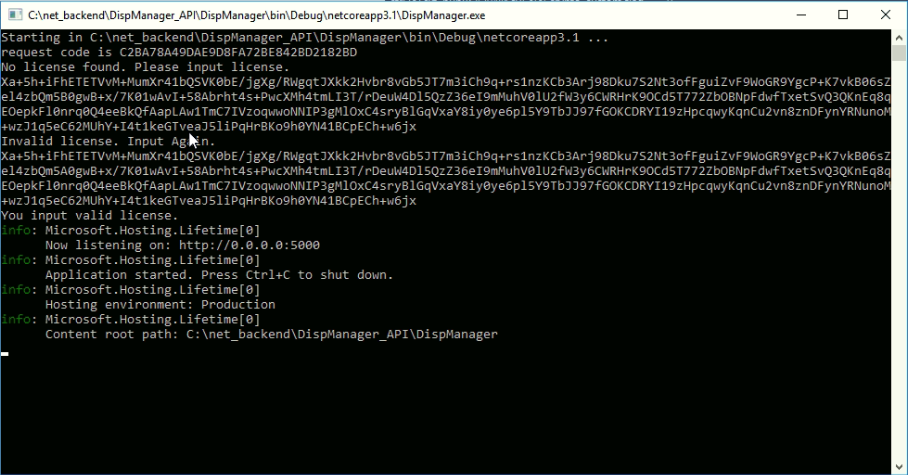
|  |  |
| --- | --- |
| UInt32 | crypto\_custom.sequence |
|  |  |

# Licensing system in .net application

## Input new activation code

At first time to use this .net application, there is no any activation code given. So a user should input activation code relevant to the local machine.

If a user run .net application, he could see the waiting state for a valid activation code like this.



Activation code that a user input from a provider given (valid)

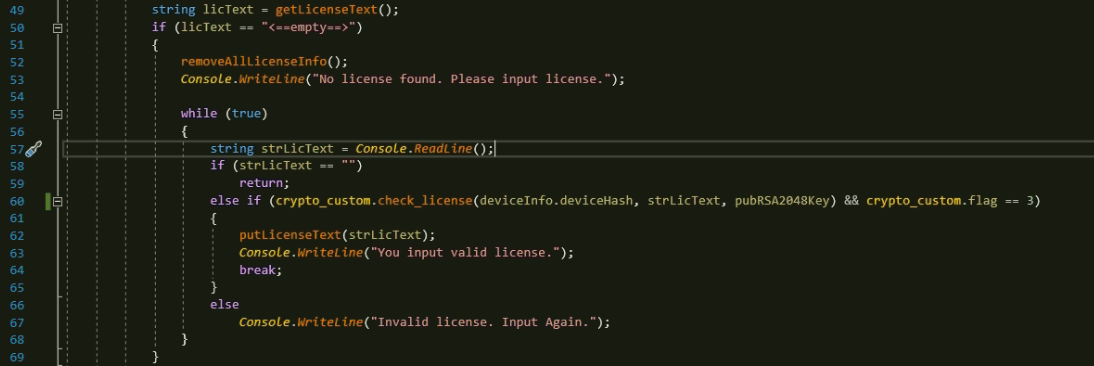
Activation code that a user input from a provider given (invalid)

Device id string for activation code

.net application

.net application is built from the project “DispManager”. Here is the code of inquiring an activation code.

If it finds no license code, then it enters waiting status for an activation code.



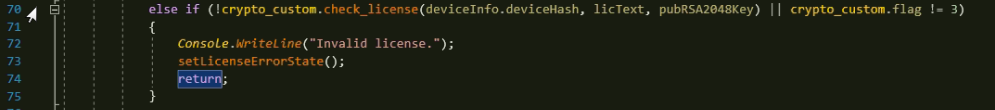
Invalid license found, input again…

Go to next step

Save a valid activation code

Waiting status for an activation code

If it finds an activation code which is already existing, then it checks.



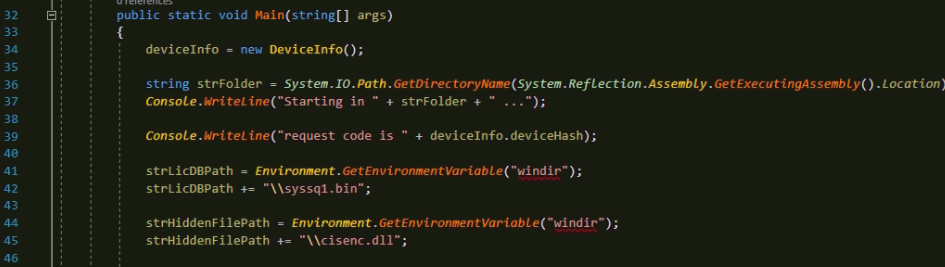
Invalid license found, remove all license information and exit…

### Save a valid activation code

If a user input a valid activation code, then we save it at local storage as an encrypted sqlite db file.

And moreover, we create a hidden file.

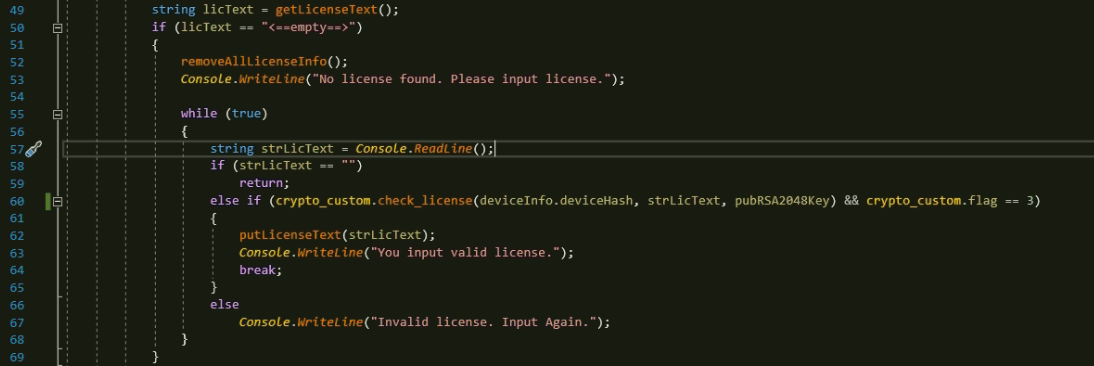
The paths to a sqlite db file and a hidden file are as follows.



Now file paths are in the table below.

|  |  |
| --- | --- |
| Encrypted sqlite db file | C:\Windows\syssq1.bin |
| Hidden file | C:\Windows\cisenc.dll |

When we input a valid activation code, we call putLicenseText() function.

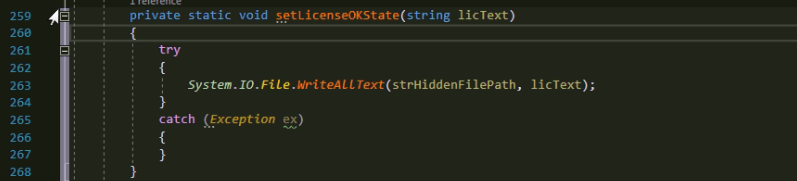


Save a valid activation code

This function creates encrypted sqlite db file and hidden file.

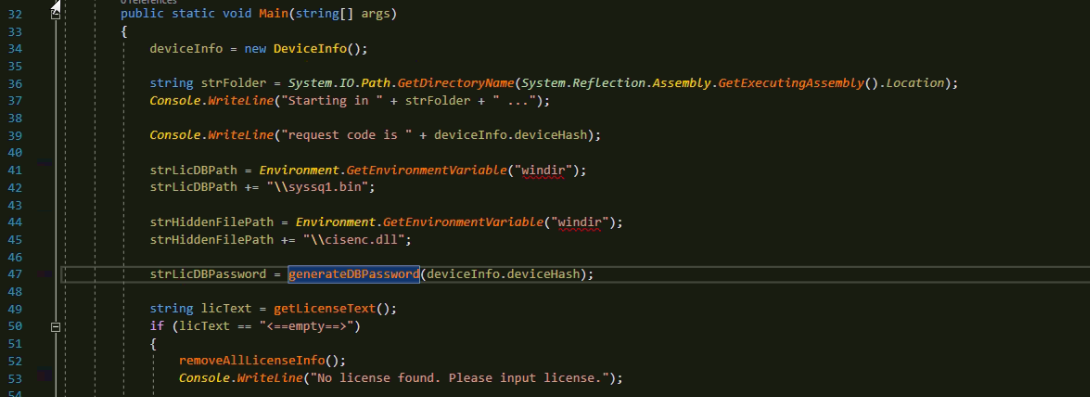
Creation of hidden file is done in setLicenseOKState() function.

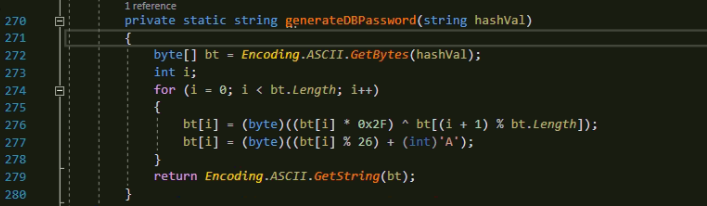
It writes the whole part of activation code to a hidden file.



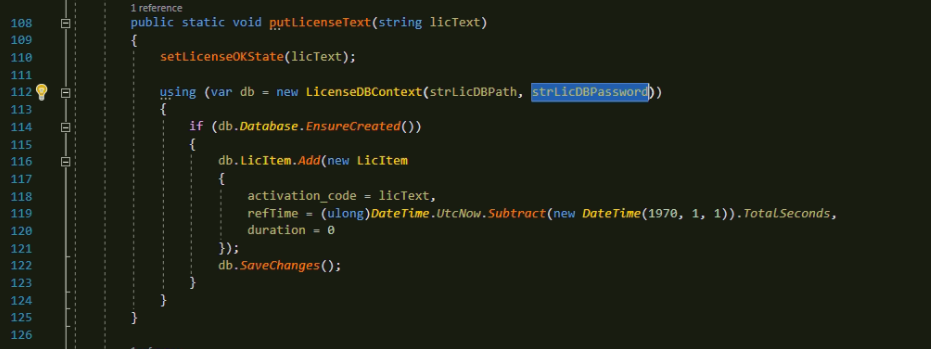
Create of encrypted sqlite db file is done in putLicenseText() function.

Sqlite db file needs a password for encryption. The password is created by generateDBPassword() function. The password is outcome of transformation of Device Id String.





We save the activation code in sqlite db file with this password.



Add a record to db.

Save to a db file with encryption.

Sqlite db password

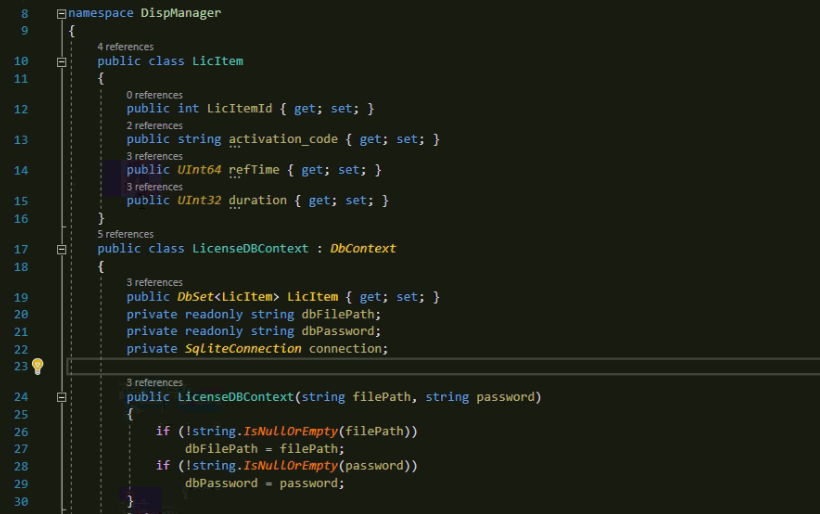
Create a db file for sure.

Here you can see the sqlite db wrapper class LicenseDBContext. This is a class that creates/reads/writes db content to/from a db file.

This class has a DbSet of LicItem that represents a table named “LicItem”.

Table “LicItem” has 4 fields in order.

|  |  |  |  |
| --- | --- | --- | --- |
| Used as a key | string that a user input | Used to check for expiration | |
| LicItemId(auto-increment key) | activation\_code | refTime | duratoin |



### To view the content of the encrypted sqlite db file.

The content of the sqilte db file can not be shown without password given, even database browsers can not view the content of the sqlite db file because they don’t support password/encryption for sqlite db.

To enable showing of encrypted sqlite db file content by a database browser, the browser should have the .dll file that supports sqlite db encryption. The individual should make .dll file on his/her own.

For example, navicat has “sqlite3.dll” to show the content of sqlite db file. At the initial installation of Navicat browser, this dll file does not support password/encryption for sqlite db file. If you want to show the content of encrypted sqlite db file by this Navicat browser, you should make “sqlite3.dll” file that supports password/encryption.

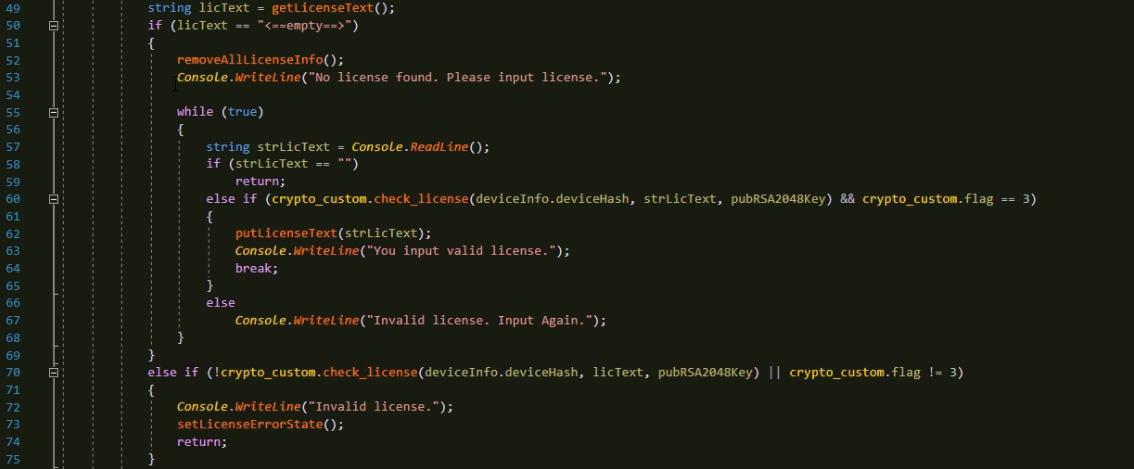
But to make sqlite3.dll file, you should know the source project that built this sqlite3.dll file. If you can get the right version of source code project, then you can add password/encryption code to the project and rebuild the project to product sqlite3.dll that supports password/encryption.

By replacing newer sqlite3.dll in Navicat installation directory, Navicat can view the content of encrypted sqlite db file.

## Check validation of the existing activation code at startup

While starting up .net application, if the encrypted sqlite db file exists, then we try to get the activation code from it and check its validation.

[Validation check is done by “crypt\_custom” class](#_Check_the_validity).



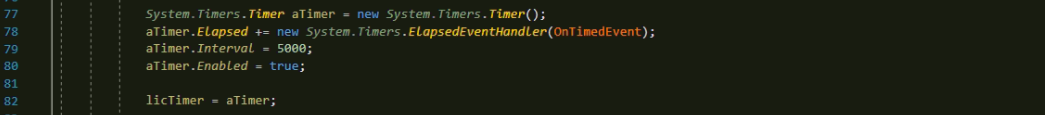
Check the validity of the activation code.

If the activation code exists in encrypted sqlite db file

## Periodic validation check of activation code

If a user inputs a valid activation code, .net application starts a timer that checks licensing information periodically.

Here is the source code of creating periodic timer.



Period in milliseconds

As you can see, timer will be timed out every 5 seconds.

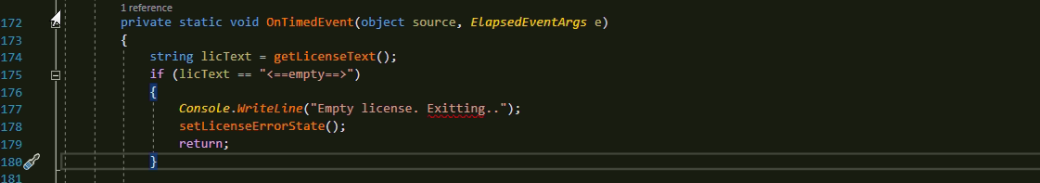
When the timer is timed out, it calls the callback function “OnTimedEvent”.

Our periodic check of license information is in OnTimedEvent function. OnTimedEvent() function works in 6 steps.

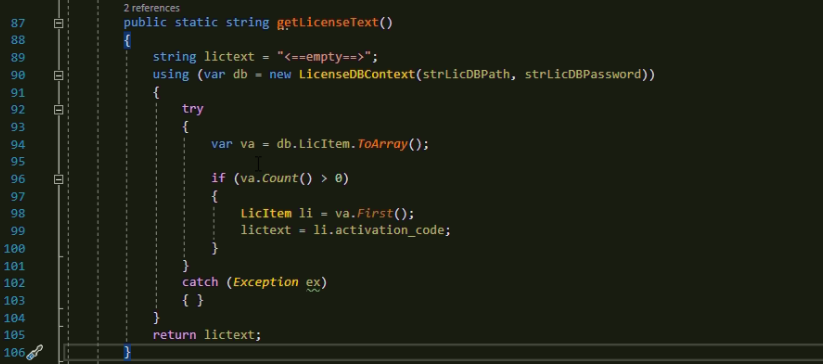
|  |
| --- |
| Get activation code from encrypted sqlite db file |
| Check its validity |
| Check its expiration |
| Check the existence of hidden file |
| Check the content of hidden file as an activation code backup. |
| Proceed hardware lock check |

### Get activation code from encrypted sqlite db file

After a user inputs a valid activation code, an encrypted sqlite db file is created. We get activation code from this encrypted sqlite db file.

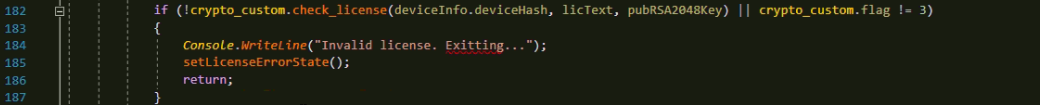


getLicenseText() function retrieves the activation code from encrypted sqlite db file that a user input.



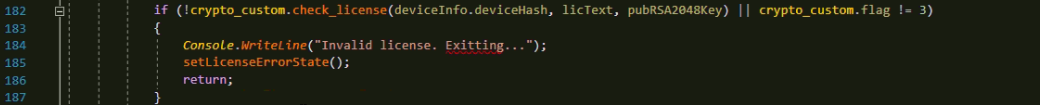
### Check the validity of the activation code retrieved from an encrypted sqlite db file.

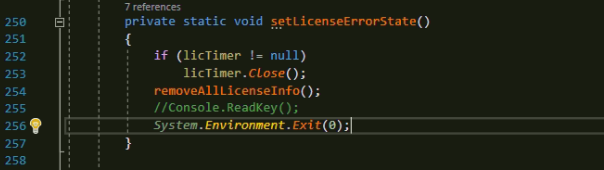
After retrieving the activation code from an encrypted sqlite db file, we check its validity.



Checking function is the one of the class “crypto\_custom” in “LicenseLib”. If checking is failed or the “flag” value is not 3, we determine that the activation code is not valid and remove all licensing information.

Removing all licensing information is done in function setLicenseErrorState().



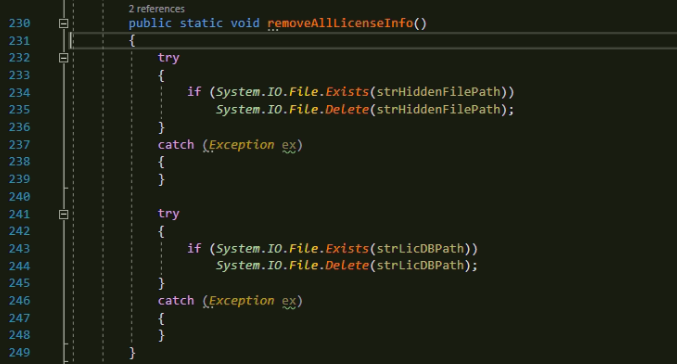


Close .net application.

Remove all licensing info.

Close timer that is running.

removeAllLicenseInfo() function removes encrypted sqlite db file and hidden file.



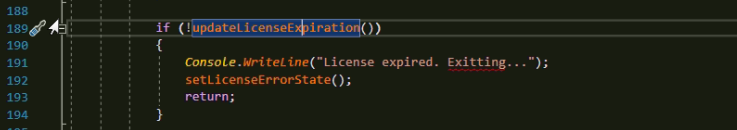
### Expiration check

OnTimedEvent() function checks the expiration of the activation code.

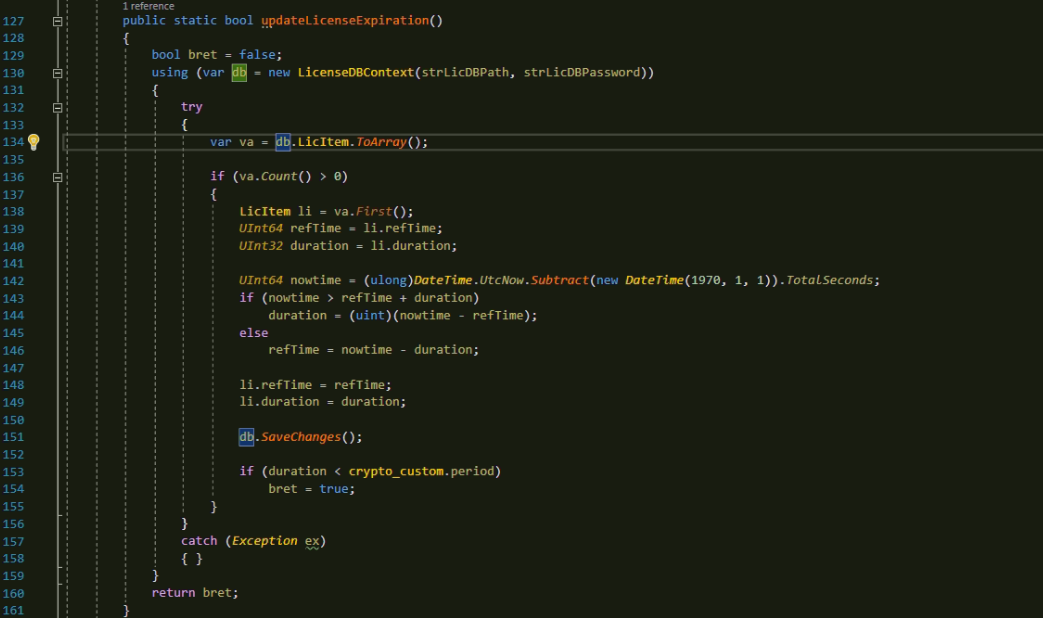
Activation code has “issued at” and “period” items. Expiration of the activation code is defined as follows.

|  |
| --- |
| “Issued at” <= now <= “issued at” + “period” |

At every timeout, “now” changes and we check the inequality defined above.



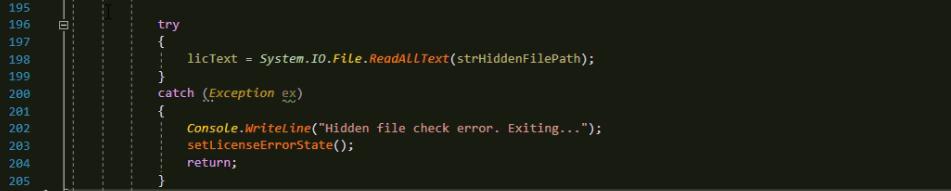
Every time we call updateLicenseExpiration(), it reads “refTime”, “duration” from an encrypted sqlite db file and checks its expiration.



### Check the existence of the hidden file

Hidden file is created when a user input a new valid activation code.

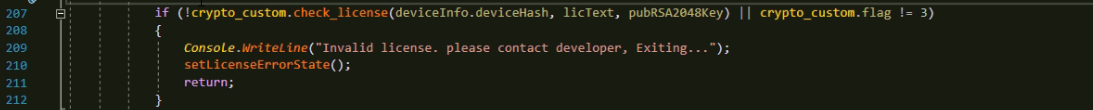
If the file does not exist at the path specified, then we determine that licensing system was corrupted and remove all licensing information.



If the hidden file does not exist

### Check the activation code of the hidden file backup

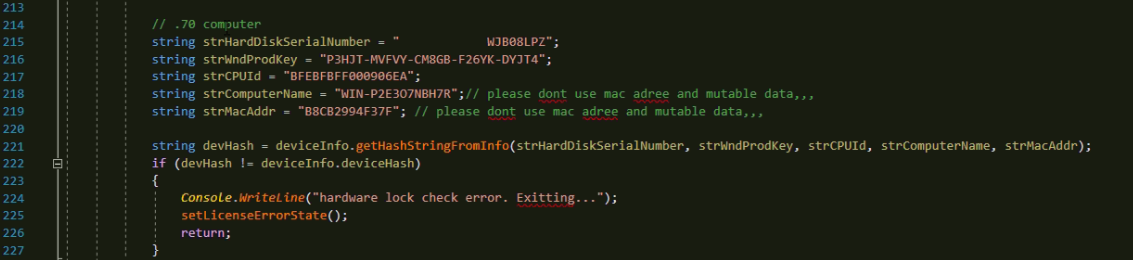
This step is like “[Check the validity of the activation code retrieved from an encrypted sqlite db file](#_Check_the_validity).”



### Hardware lock check

Hardware lock check is a kind of restriction of running platform.

It compares the device id string with a constant one that is fixed by a provider.



getHashStringFromInfo() function implements calculation of device id string so that it determines [what kind of hardwares are taken part in the hardware lock check](#_How_to_bar).

# External resources used for .net licensing

## Folder

No folder used.

## File

|  |  |
| --- | --- |
| Encrypted sqlite db file | C:\Windows\syssq1.bin |
| Hidden file | C:\Windows\cisenc.dll |

## Registry

The registry key “HKEY\_LOCAL\_MACHINE/SOFTWARE/Microsoft/Windows NT/CurrentVersion” is used for retrieving [windows product id](#_Windows_Product_Id).

## Hardware

|  |  |
| --- | --- |
| Used | Hard disk serial number |
| Used | Windows Product id |
| Used | CPU id |
| Not used | Computer Name |
| Not used | MAC address |

# FAQ

#### [How to bar hardware information from composition of device id string](#_How_to_bar)

#### [How to change the period of license check](#_Periodic_validation_check)