.net Licensing Manual

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

1. Device Information

Licensing is initiated from getting unique machine information.

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

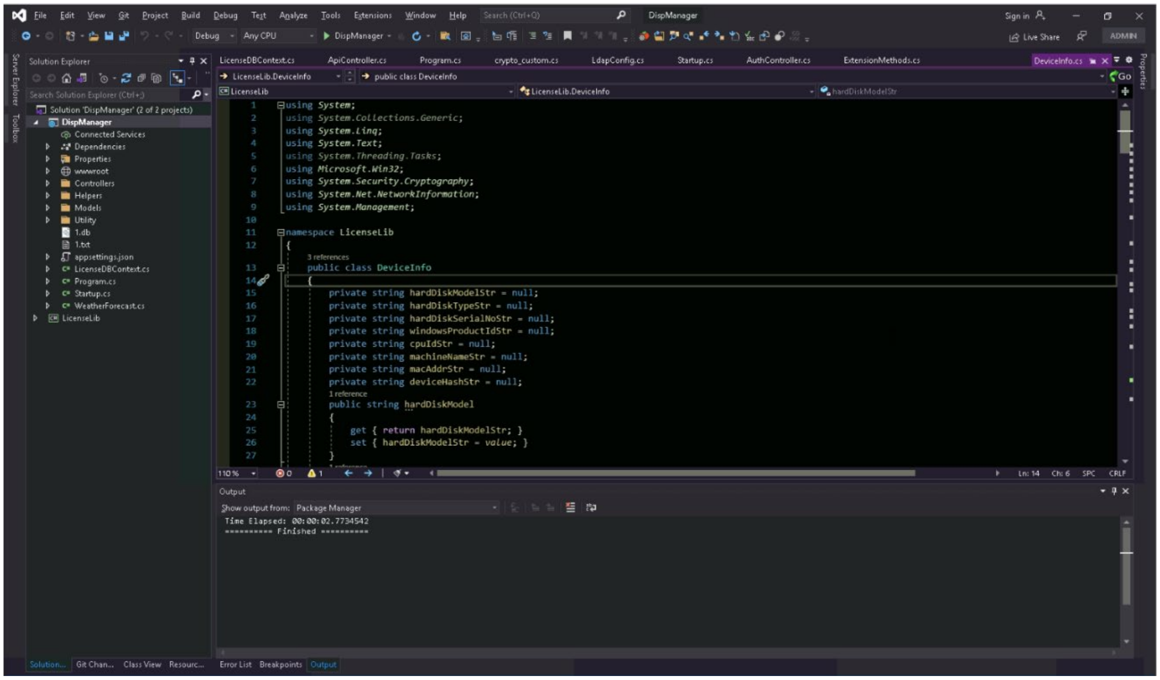
Device information is composed of the following information.

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”.

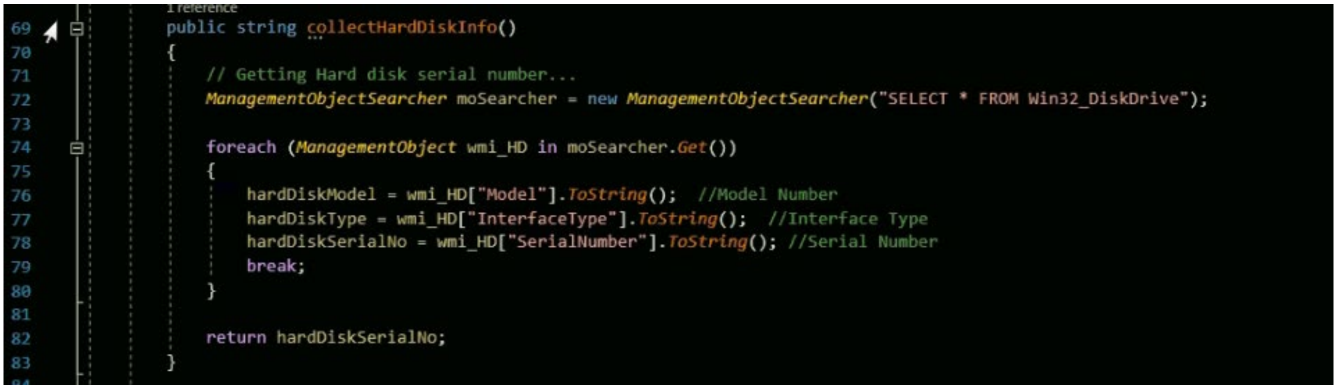


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.Win32;
using System.Security.Cryptography;
using System.Net.NetworkInformation;
using System.Management;

namespace LicenseLib
{
    public class DeviceInfo
    {
        private string hardDiskModelStr = null;
        private string hardDiskTypeStr = null;
        private string hardDiskSerialNoStr = null;
        private string windowsProductIdStr = null;
        private string cpuIdStr = null;
        private string machineNameStr = null;
        private string macAddrStr = null;
        private string deviceHashStr = null;
        public string hardDiskModel
        {
            get { return hardDiskModelStr; }
            set { hardDiskModelStr = value; }
        }
```

```
Time Elapsed: 00:00:02.7734542
========== Finished ==========
```

1) Hard disk serial number

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



```
1 reference
public string collectHardDiskInfo()
{
    // Getting Hard disk serial number...
    ManagementObjectSearcher moSearcher = new ManagementObjectSearcher("SELECT * FROM Win32_DiskDrive");

    foreach (ManagementObject wmi_HD in moSearcher.Get())
    {
        hardDiskModel = wmi_HD["Model"].ToString();  //Model Number
        hardDiskType = wmi_HD["InterfaceType"].ToString();  //Interface Type
        hardDiskSerialNo = wmi_HD["SerialNumber"].ToString(); //Serial Number
        break;
    }

    return hardDiskSerialNo;
}
```

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”

2) Windows Product Id

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

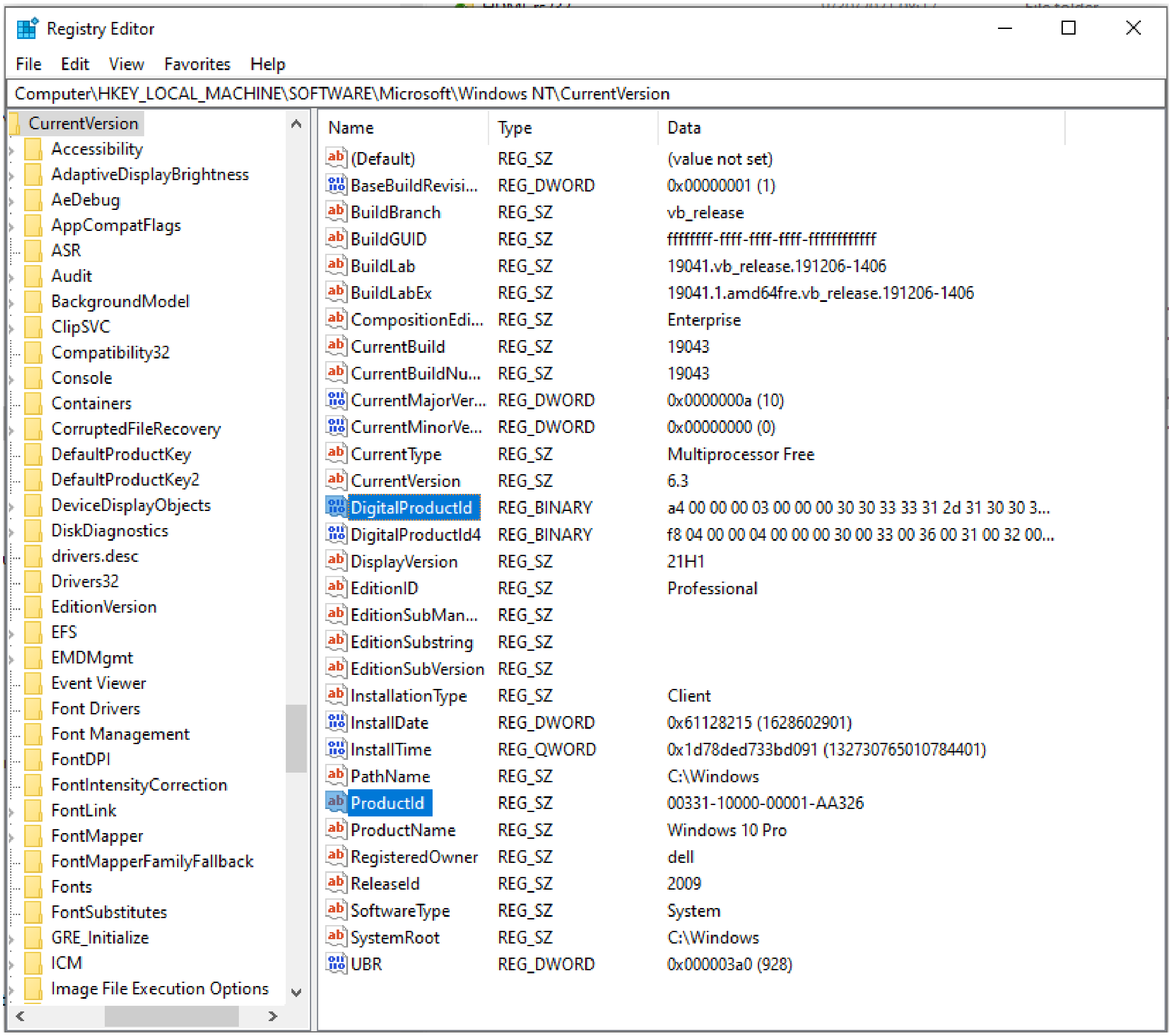
It can be retrieved from the registry key.

“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.



Computer\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion

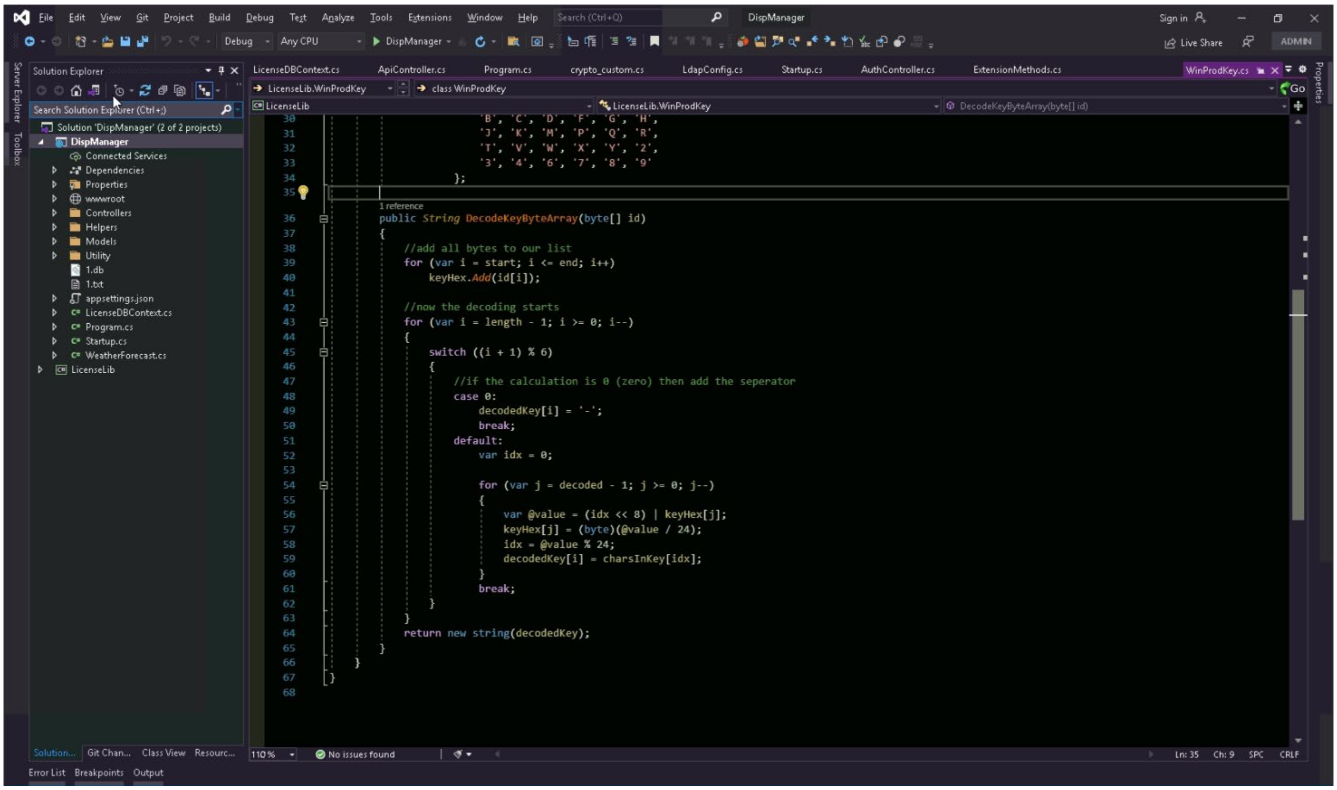
Name	Type	Data
(Default)	REG_SZ	(value not set)
BaseBuildRevisi...	REG_DWORD	0x00000001 (1)
BuildBranch	REG_SZ	vb_release
BuildGUID	REG_SZ	ffffffff-ffff-ffff-ffff-ffffffffffff
BuildLab	REG_SZ	19041.vb_release.191206-1406
BuildLabEx	REG_SZ	19041.1.amd64fre.vb_release.191206-1406
CompositionEdi...	REG_SZ	Enterprise
CurrentBuild	REG_SZ	19043
CurrentBuildNu...	REG_SZ	19043
CurrentMajorVer...	REG_DWORD	0x0000000a (10)
CurrentMinorVe...	REG_DWORD	0x00000000 (0)
CurrentType	REG_SZ	Multiprocessor Free
CurrentVersion	REG_SZ	6.3
DigitalProductId	REG_BINARY	a4 00 00 00 03 00 00 00 30 30 33 33 31 2d 31 30 30 3...
DigitalProductId4	REG_BINARY	f8 04 00 00 04 00 00 00 30 00 33 00 36 00 31 00 32 00...
DisplayVersion	REG_SZ	21H1
EditionID	REG_SZ	Professional
EditionSubMan...	REG_SZ	
EditionSubstring	REG_SZ	
EditionSubVersion	REG_SZ	
InstallationType	REG_SZ	Client
InstallDate	REG_DWORD	0x61128215 (1628602901)
InstallTime	REG_QWORD	0x1d78ded733bd091 (132730765010784401)
PathName	REG_SZ	C:\Windows
ProductId	REG_SZ	00331-10000-00001-AA326
ProductName	REG_SZ	Windows 10 Pro
RegisteredOwner	REG_SZ	dell
ReleaseId	REG_SZ	2009
SoftwareType	REG_SZ	System
SystemRoot	REG_SZ	C:\Windows
UBR	REG_DWORD	0x000003a0 (928)

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.

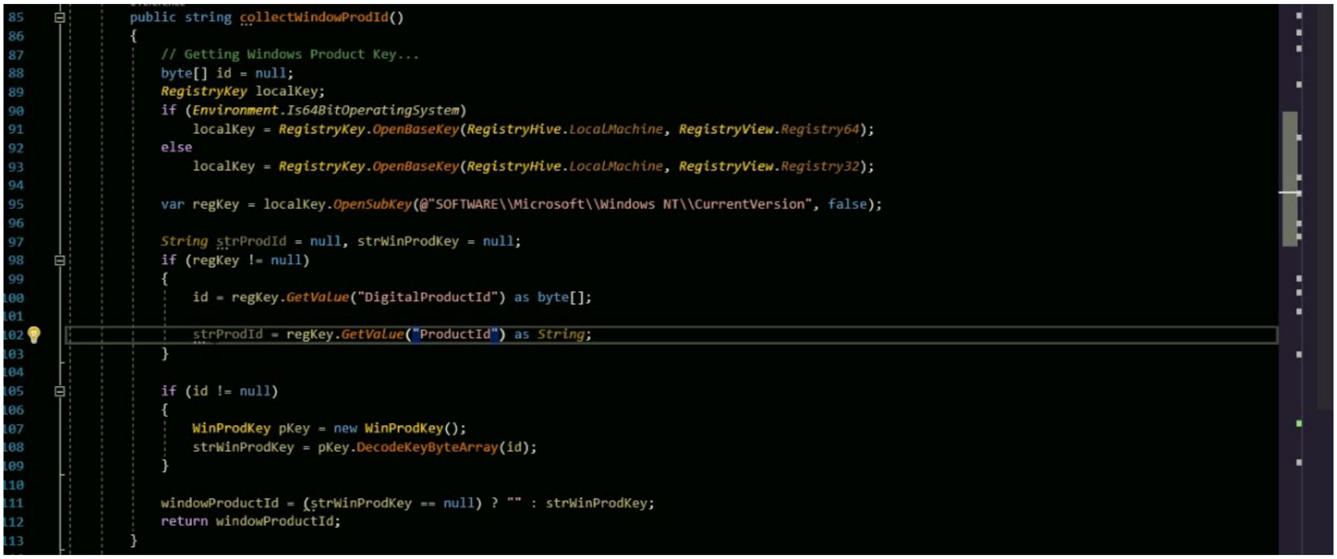
Conversion is done by “WinProdKey” class.

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



```
                'B', 'C', 'D', 'F', 'G', 'H',
                'J', 'K', 'M', 'P', 'Q', 'R',
                'T', 'V', 'W', 'X', 'Y', '2',
                '3', '4', '6', '7', '8', '9'
            };

        1 reference
        public String DecodeKeyByteArray(byte[] id)
        {
            //add all bytes to our list
            for (var i = start; i <= end; i++)
                keyHex.Add(id[i]);

            //now the decoding starts
            for (var i = length - 1; i >= 0; i--)
            {
                switch ((i + 1) % 6)
                {
                    //if the calculation is 0 (zero) then add the seperator
                    case 0:
                        decodedKey[i] = '-';
                        break;
                    default:
                        var idx = 0;

                        for (var j = decoded - 1; j >= 0; j--)
                        {
                            var @value = (idx << 8) | keyHex[j];
                            keyHex[j] = (byte)(@value / 24);
                            idx = @value % 24;
                            decodedKey[i] = charsInKey[idx];
                        }
                        break;
                }
            }
            return new string(decodedKey);
        }
    }
}
```

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



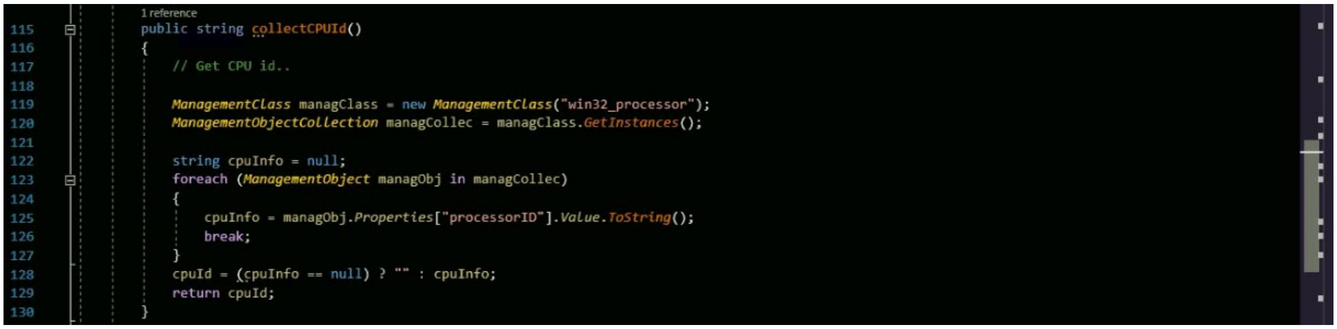
```
public string collectWindowProdId()
{
    // Getting Windows Product Key...
    byte[] id = null;
    RegistryKey localKey;
    if (Environment.Is64BitOperatingSystem)
        localKey = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry64);
    else
        localKey = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, RegistryView.Registry32);

    var regKey = localKey.OpenSubKey(@"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion", false);

    String strProdId = null, strWinProdKey = null;
    if (regKey != null)
    {
        id = regKey.GetValue("DigitalProductId") as byte[];

        strProdId = regKey.GetValue("ProductId") as String;
    }

    if (id != null)
    {
        WinProdKey pKey = new WinProdKey();
        strWinProdKey = pKey.DecodeKeyByteArray(id);
    }

    windowProductId = (strWinProdKey == null) ? "" : strWinProdKey;
    return windowProductId;
}
```

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

3) CPU id

CPU id is a string representing the unique CPU identifier.

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

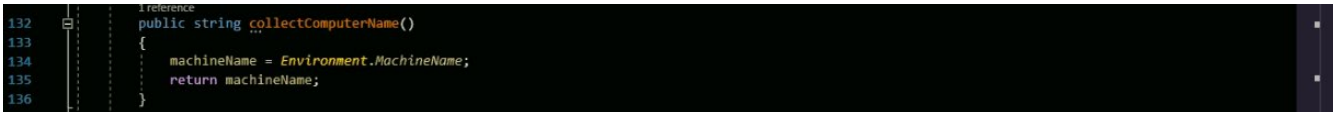


```
public string collectCPUId()
{
    // Get CPU id..

    ManagementClass managClass = new ManagementClass("win32_processor");
    ManagementObjectCollection managCollec = managClass.GetInstances();

    string cpuInfo = null;
    foreach (ManagementObject managObj in managCollec)
    {
        cpuInfo = managObj.Properties["processorID"].Value.ToString();
        break;
    }
    cpuId = (cpuInfo == null) ? "" : cpuInfo;
    return cpuId;
}
```

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

4) 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.



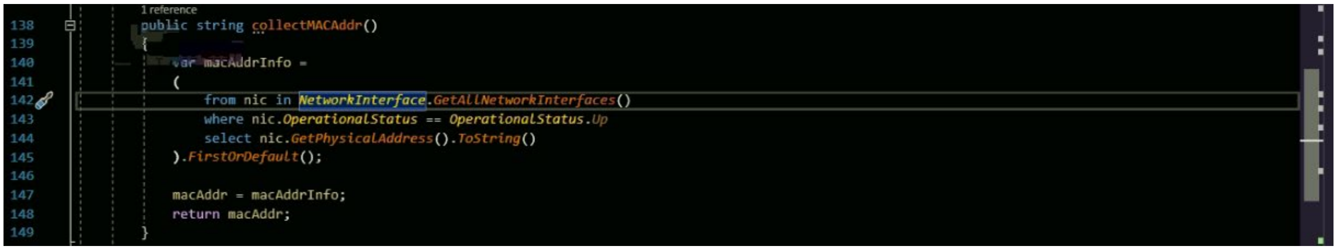
```
public string collectComputerName()
{
    machineName = Environment.MachineName;
    return machineName;
}
```

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

5) 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.



```
public string collectMACAddr()
{
    var macAddrInfo =
    (
        from nic in NetworkInterface.GetAllNetworkInterfaces()
        where nic.OperationalStatus == OperationalStatus.Up
        select nic.GetPhysicalAddress().ToString()
    ).FirstOrDefault();

    macAddr = macAddrInfo;
    return macAddr;
}
```

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.

6) 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.

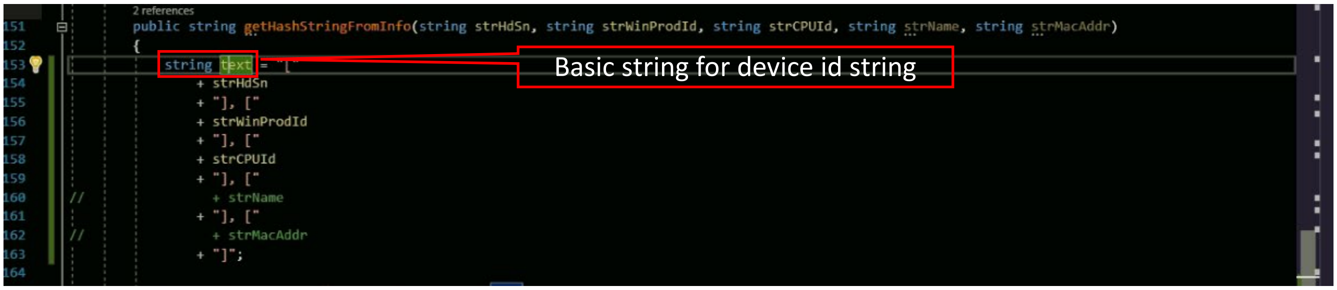
The fourth parameter “strName” is the Computer Name.

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.

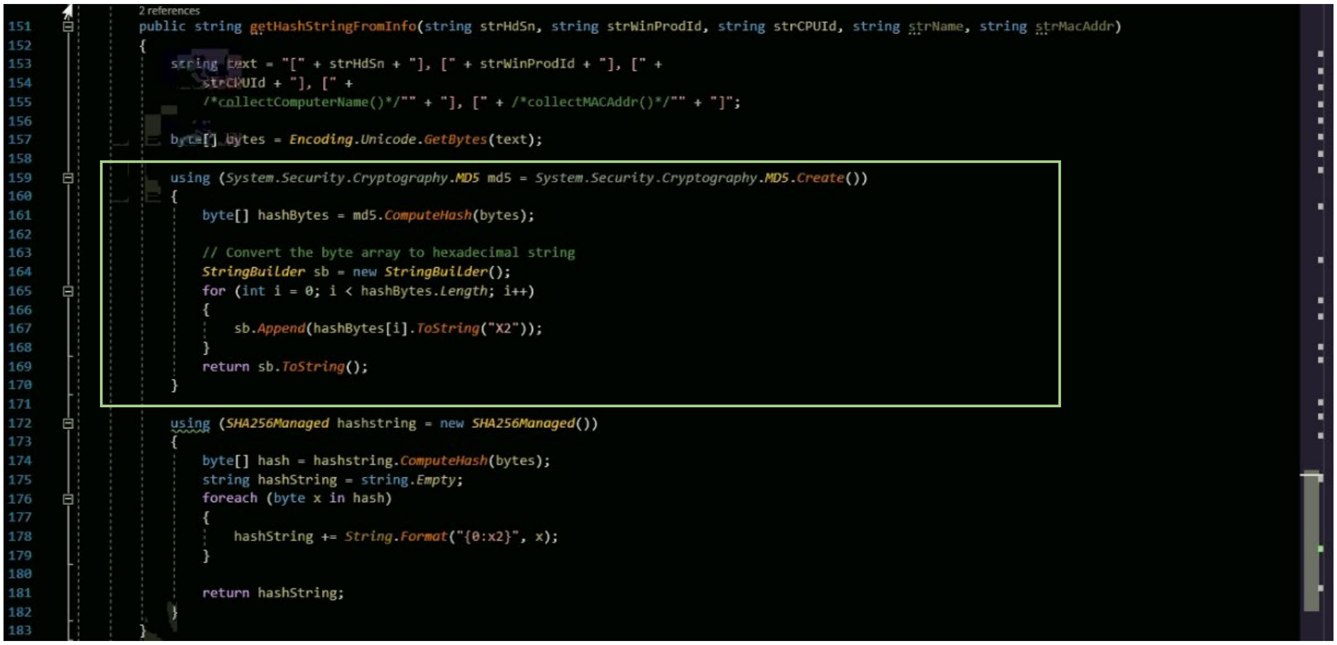


```
        2 references
151     public string getHashStringFromInfo(string strHdSn, string strWinProdId, string strCPUId, string strName, string strMacAddr)
152     {
153         string text = "["
154             + strHdSn
155             + "], ["
156             + strWinProdId
157             + "], ["
158             + strCPUId
159             + "], ["
160 //            + strName
161             + "], ["
162 //            + strMacAddr
163             + "]";
164
```

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)

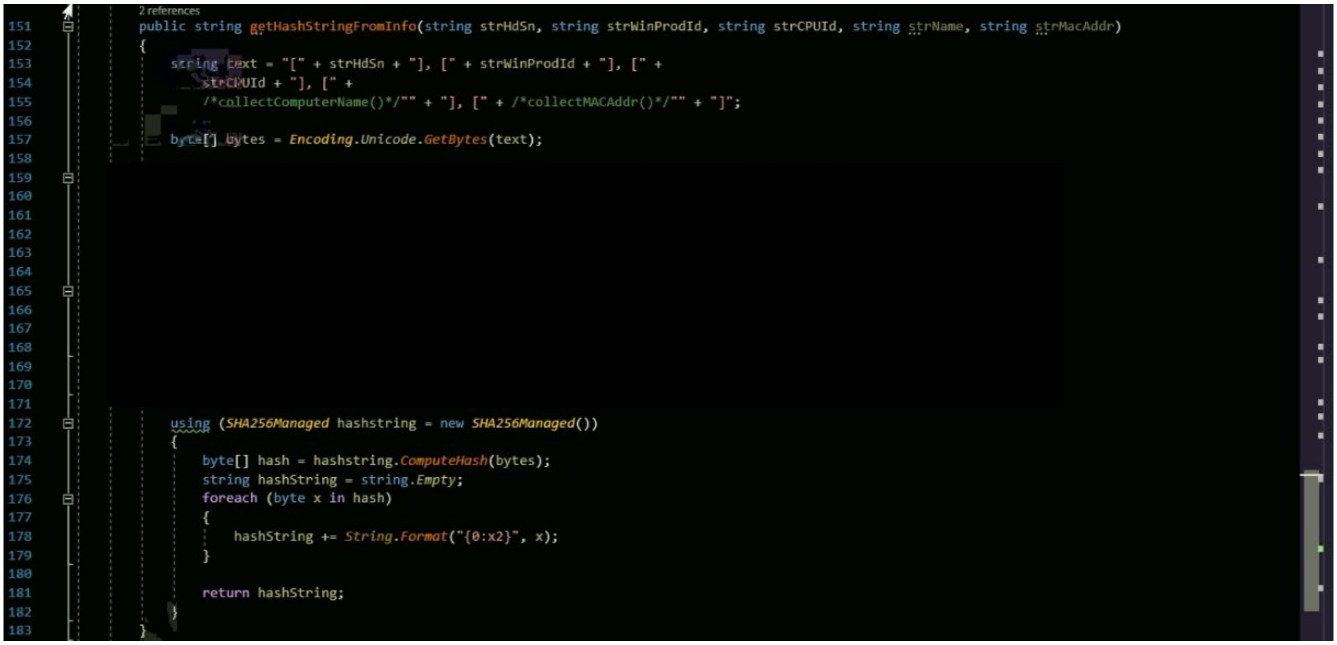
We might use MD5 or SHA256 in this function.



```
2 references
public string getHashStringFromInfo(string strHdSn, string strWinProdId, string strCPUId, string strName, string strMacAddr)
{
    string text = "[" + strHdSn + "], [" + strWinProdId + "], [" +
        strCPUId + "], [" +
        /*collectComputerName()*/"" + "], [" + /*collectMACAddr()*/"" + "]";

    byte[] bytes = Encoding.Unicode.GetBytes(text);

    using (System.Security.Cryptography.MD5 md5 = System.Security.Cryptography.MD5.Create())
    {
        byte[] hashBytes = md5.ComputeHash(bytes);

        // Convert the byte array to hexadecimal string
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < hashBytes.Length; i++)
        {
            sb.Append(hashBytes[i].ToString("X2"));
        }
        return sb.ToString();
    }

    using (SHA256Managed hashstring = new SHA256Managed())
    {
        byte[] hash = hashstring.ComputeHash(bytes);
        string hashString = string.Empty;
        foreach (byte x in hash)
        {
            hashString += String.Format("{0:x2}", x);
        }

        return hashString;
    }
}
```

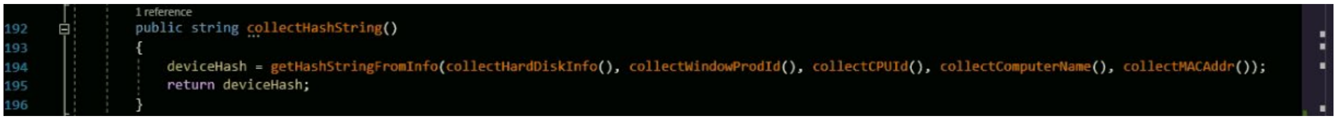
We calculate hash string by MD5 library.

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



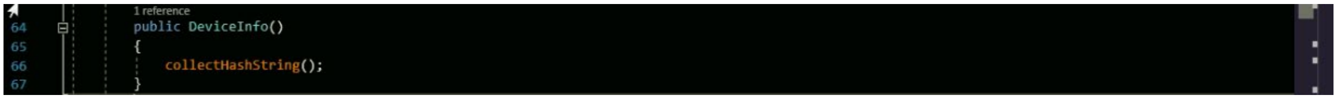
```
2 references
public string getHashStringFromInfo(string strHdSn, string strWinProdId, string strCPUId, string strName, string strMacAddr)
{
    string text = "[" + strHdSn + "], [" + strWinProdId + "], [" +
        strCPUId + "], [" +
        /*collectComputerName()*/"" + "], [" + /*collectMACAddr()*/"" + "]";

    byte[] bytes = Encoding.Unicode.GetBytes(text);

[redacted]

    using (SHA256Managed hashstring = new SHA256Managed())
    {
        byte[] hash = hashstring.ComputeHash(bytes);
        string hashString = string.Empty;
        foreach (byte x in hash)
        {
            hashString += String.Format("{0:x2}", x);
        }

        return hashString;
    }
}
```

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



```
1 reference
public string collectHashString()
{
    deviceHash = getHashStringFromInfo(collectHardDiskInfo(), collectWindowProdId(), collectCPUId(), collectComputerName(), collectMACAddr());
    return deviceHash;
}
```

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

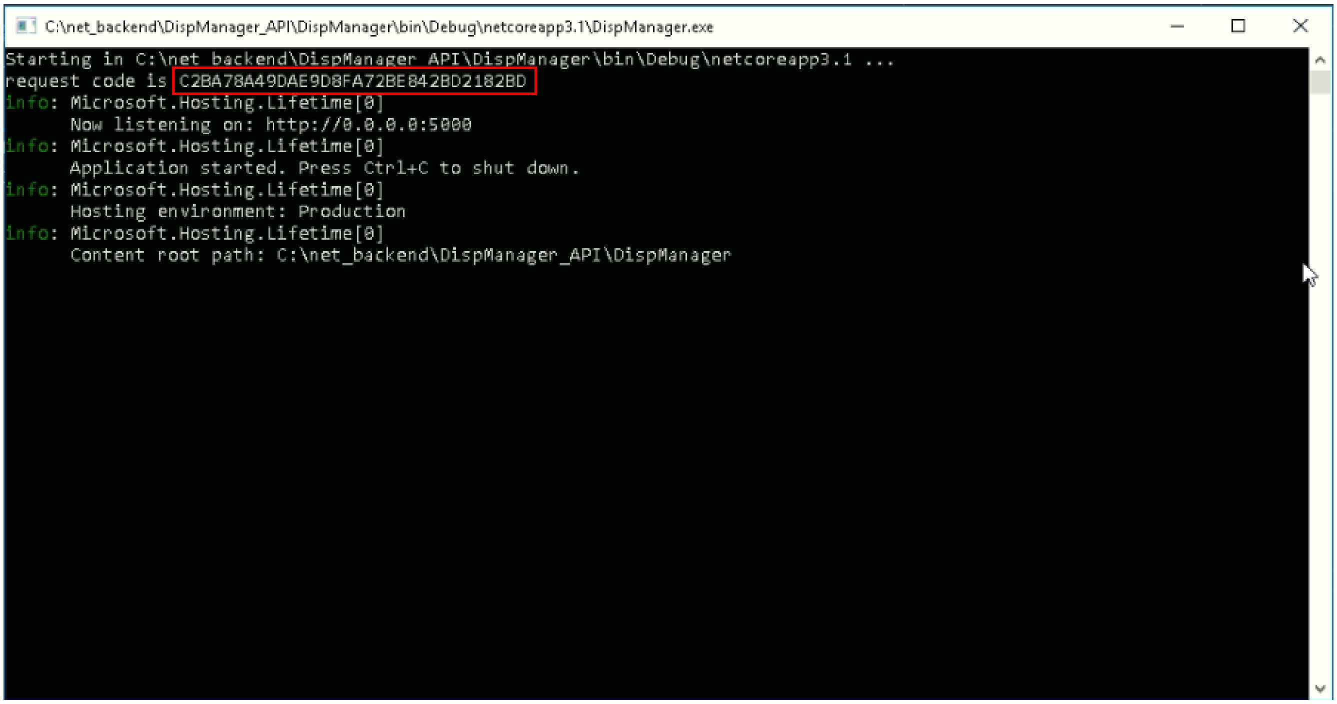


```
1 reference
public DeviceInfo()
{
    collectHashString();
}
```

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.

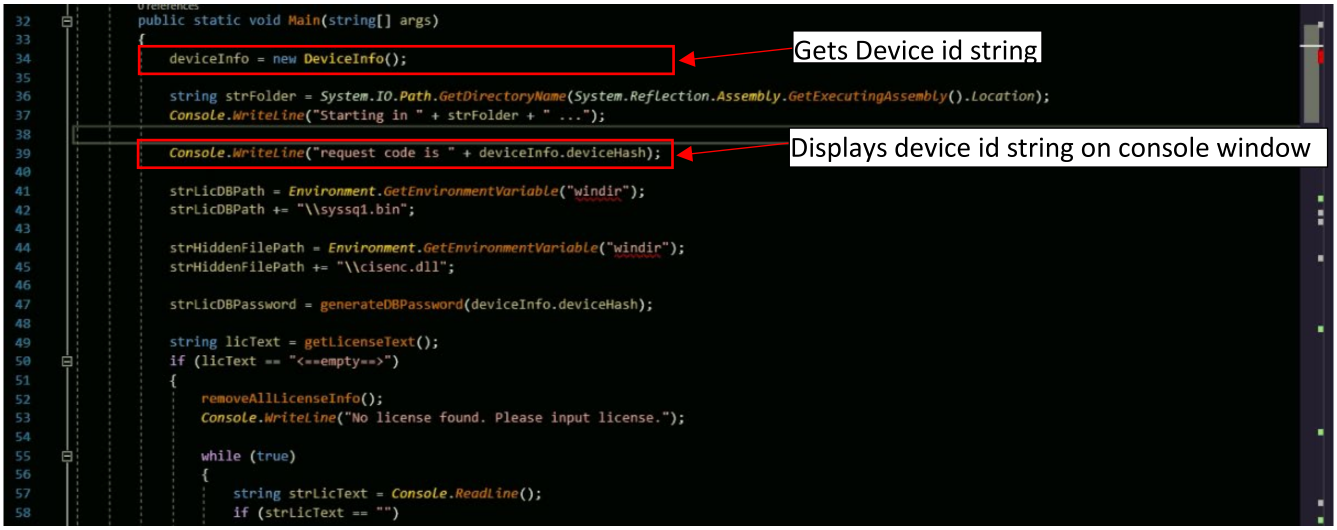


C:\net_backend\DispManager_API\DispManager\bin\Debug\netcoreapp3.1\DispManager.exe

```
Starting in C:\net_backend\DispManager_API\DispManager\bin\Debug\netcoreapp3.1 ...
request code is C2BA78A49DAE9D8FA72BE842BD2182BD
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://0.0.0.0:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\net_backend\DispManager_API\DispManager
```

This is the device information we got.

We coded it in .net project as



```
0 references
public static void Main(string[] args)
{
    deviceInfo = new DeviceInfo();

    string strFolder = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
    Console.WriteLine("Starting in " + strFolder + " ...");

    Console.WriteLine("request code is " + deviceInfo.deviceHash);

    strLicDBPath = Environment.GetEnvironmentVariable("windir");
    strLicDBPath += "\\syssq1.bin";

    strHiddenFilePath = Environment.GetEnvironmentVariable("windir");
    strHiddenFilePath += "\\cisenc.dll";

    strLicDBPassword = generateDBPassword(deviceInfo.deviceHash);

    string licText = getLicenseText();
    if (licText == "<==empty==>")
    {
        removeAllLicenseInfo();
        Console.WriteLine("No license found. Please input license.");

        while (true)
        {
            string strLicText = Console.ReadLine();
            if (strLicText == "")
```

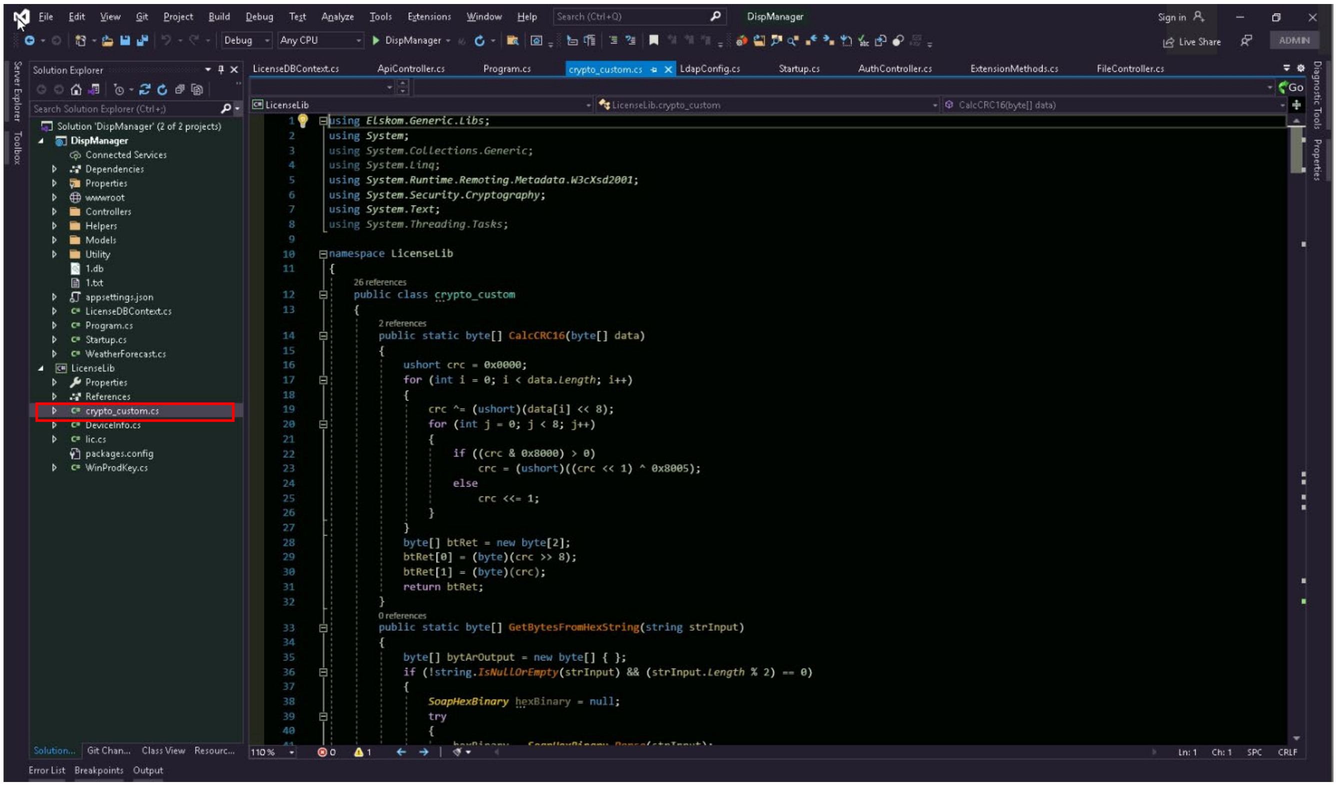
Gets Device id string

Displays device id string on console window

2. Licensing algorithm

Licensing algorithm is similar to Cpp project.

Basic functions are defined in “crypto_custom” class of LicenseLib project.



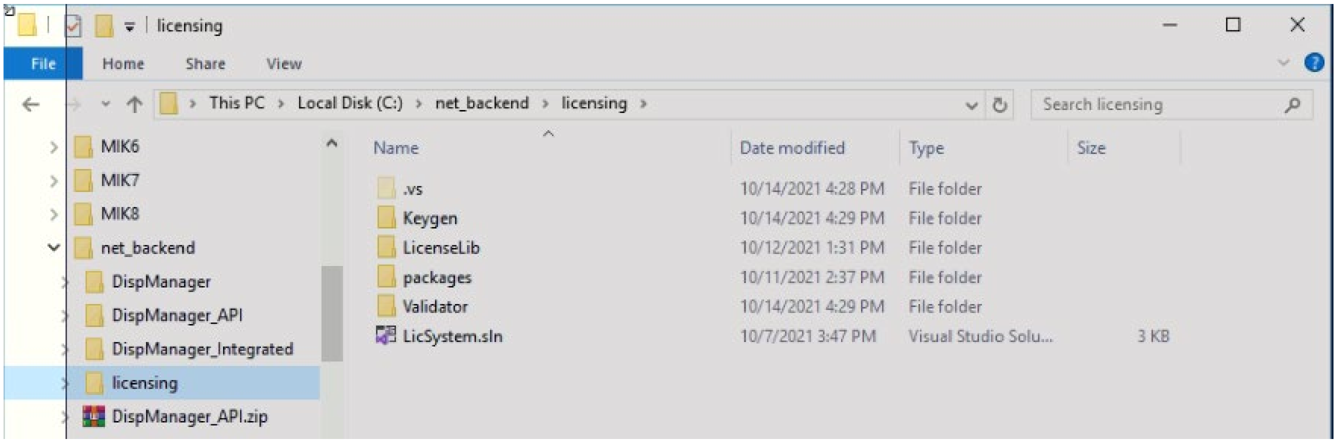
```
using Elskom.Generic.Libs;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Remoting.Metadata.W3cXsd2001;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace LicenseLib
{
    public class crypto_custom
    {
        public static byte[] CalcCRC16(byte[] data)
        {
            ushort crc = 0x0000;
            for (int i = 0; i < data.Length; i++)
            {
                crc ^= (ushort)(data[i] << 8);
                for (int j = 0; j < 8; j++)
                {
                    if ((crc & 0x8000) > 0)
                        crc = (ushort)((crc << 1) ^ 0x8005);
                    else
                        crc <<= 1;
                }
            }
            byte[] btRet = new byte[2];
            btRet[0] = (byte)(crc >> 8);
            btRet[1] = (byte)(crc);
            return btRet;
        }
        public static byte[] GetBytesFromHexString(string strInput)
        {
            byte[] bytArOutput = new byte[] { };
            if (!string.IsNullOrEmpty(strInput) && (strInput.Length % 2) == 0)
            {
                SoapHexBinary hexBinary = null;
                try
                {
```

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.

1) Generation of activation code

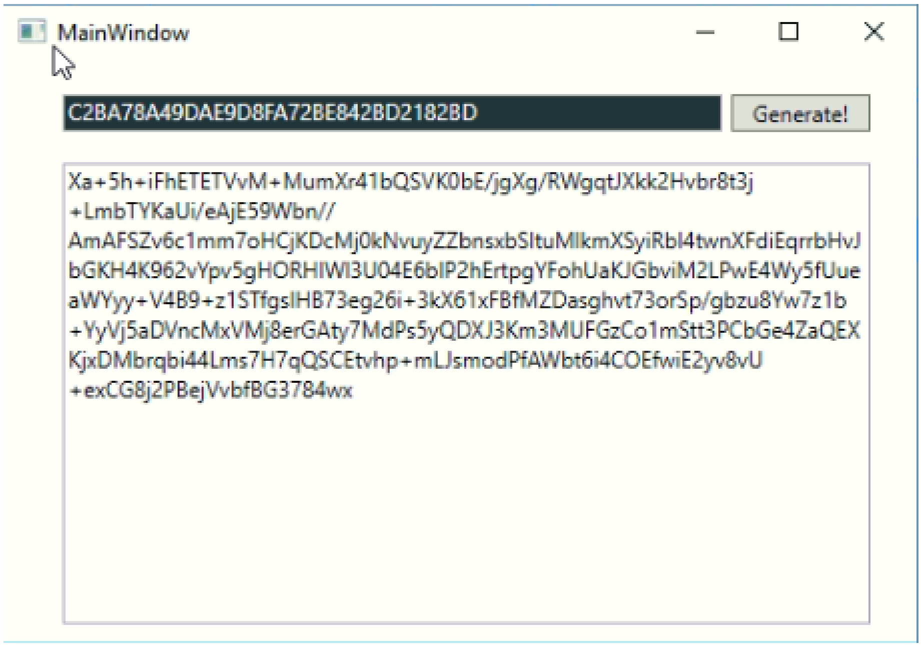
Generation of the activation code is done in keygen.exe. This application is built from the project “Keygen”.



This PC › Local Disk (C:) › net_backend › licensing ›

Name	Date modified	Type	Size
.vs	10/14/2021 4:28 PM	File folder	
Keygen	10/14/2021 4:29 PM	File folder	
LicenseLib	10/12/2021 1:31 PM	File folder	
packages	10/11/2021 2:37 PM	File folder	
Validator	10/14/2021 4:29 PM	File folder	
LicSystem.sln	10/7/2021 3:47 PM	Visual Studio Solu...	3 KB

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

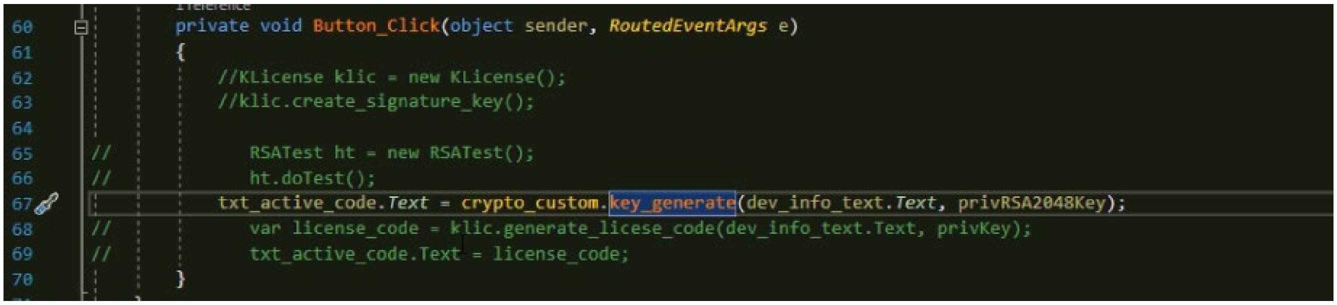


C2BA78A49DAE9D8FA72BE842BD2182BD

```
Xa+5h+iFhETETVvM+MumXr41bQSVK0bE/jgXg/RWgqtJXkk2Hvbr8t3j
+LmbTYKaUi/eAjE59Wbn//
AmAFSZv6c1mm7oHCjKDcMj0kNvuyZZbnsxbSltuMlkmXSyiRbl4twnXFdiEqrrbHvJ
bGKH4K962vYpv5gHORHIWl3U04E6bIP2hErtpgYFohUaKJGbviM2LPwE4Wy5fUue
aWYyy+V4B9+z1STfgsIHB73eg26i+3kX61xFBfMZDasghvt73orSp/gbzu8Yw7z1b
+YyVj5aDVncMxVMj8erGAty7MdPs5yQDXJ3Km3MUFGzCo1mStt3PCbGe4ZaQEX
KjxDMbrqbi44Lms7H7qQSCEtvhp+mLJsmodPfAWbt6i4COEfwiE2yv8vU
+exCG8j2PBejVvbfBG3784wx
```

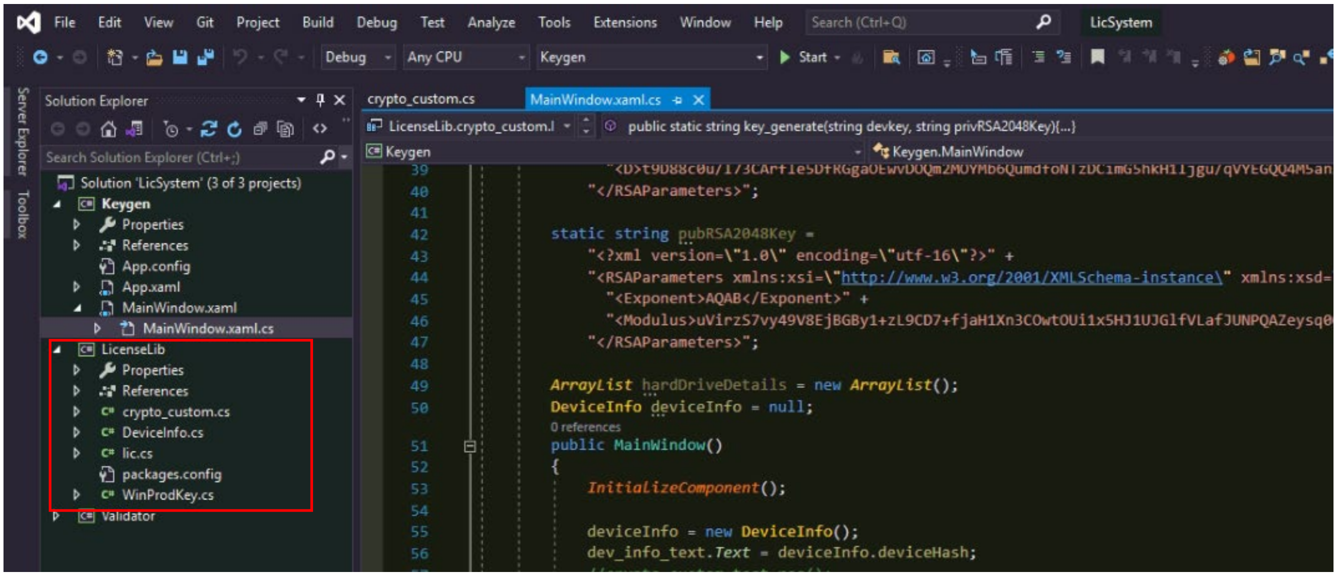
The project “Keygen” is very simple project which makes activation code based on the project “LicenseLib”.

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



```
    1 reference
    private void Button_Click(object sender, RoutedEventArgs e)
    {
        //KLicense klic = new KLicense();
        //klic.create_signature_key();

//          RSATest ht = new RSATest();
//          ht.doTest();
        txt_active_code.Text = crypto_custom.key_generate(dev_info_text.Text, privRSA2048Key);
//          var license_code = klic.generate_licese_code(dev_info_text.Text, privKey);
//          txt_active_code.Text = license_code;
    }
```

Here calls the key_generate() function of the class “crypto_custom”. This class is the one of “LicenseLib” project.



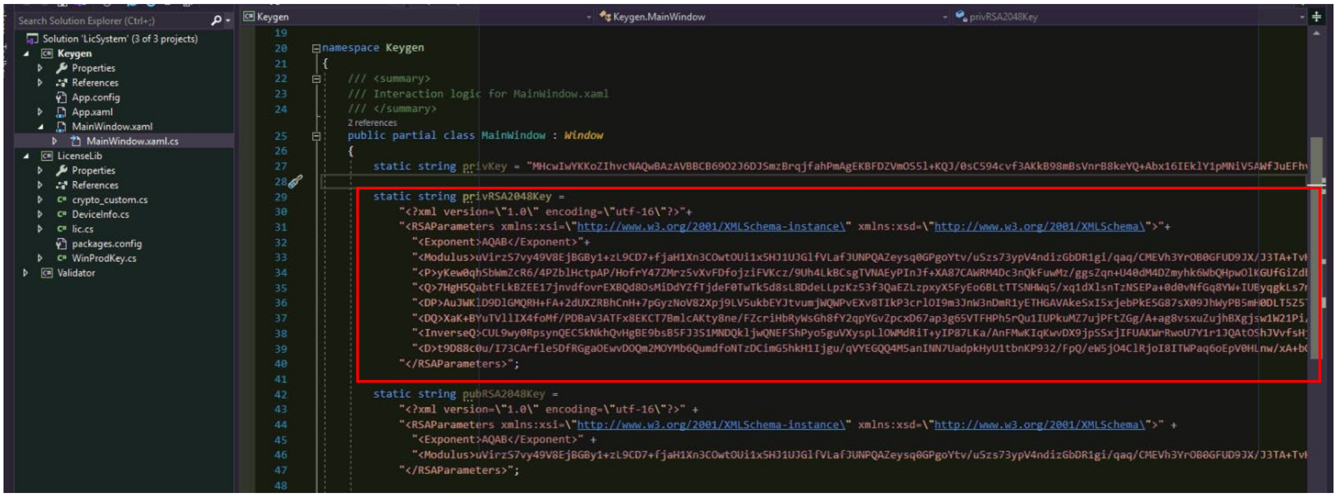
```
            "<D>t9D88c0u/173CArf1e5DfRGgaOEwvDOQm2MOYMb6QumdfoNTzDCimG5hKH11jgu/qVYEGQQ4M5an
            "</RSAParameters>";

        static string pubRSA2048Key =
            "<?xml version=\"1.0\" encoding=\"utf-16\"?>" +
            "<RSAParameters xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=
              "<Exponent>AQAB</Exponent>" +
              "<Modulus>uVirzS7vy49V8EjBGBy1+zL9CD7+fjaH1Xn3COwtOUi1x5HJ1UJGlfVLafJUNPQAZeysq0
            "</RSAParameters>";

        ArrayList hardDriveDetails = new ArrayList();
        DeviceInfo deviceInfo = null;
        0 references
        public MainWindow()
        {
            InitializeComponent();

            deviceInfo = new DeviceInfo();
            dev_info_text.Text = deviceInfo.deviceHash;
```

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

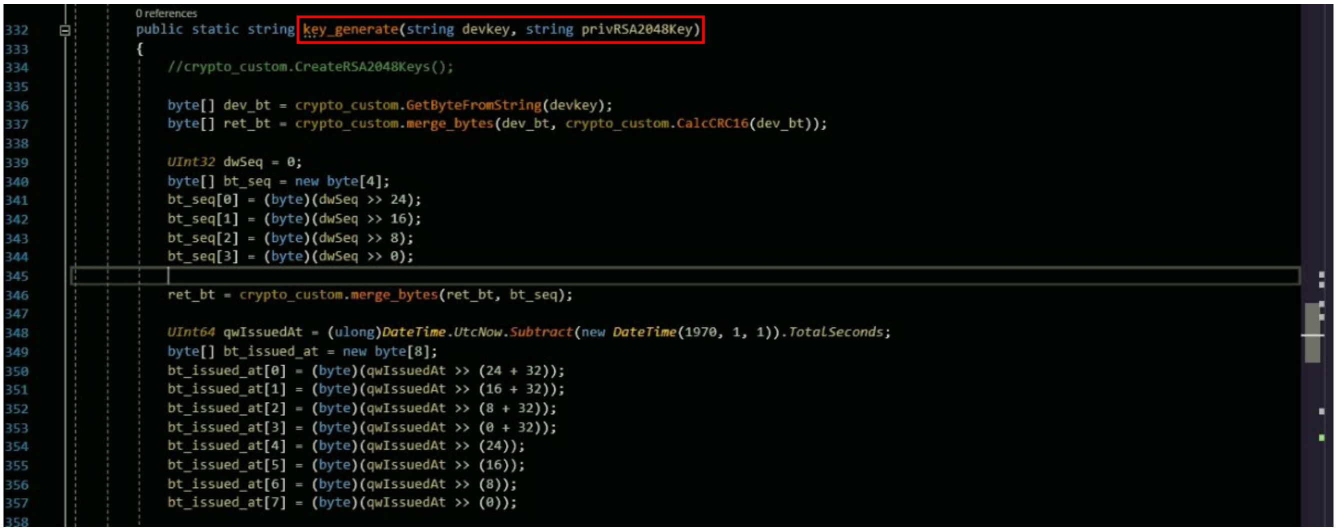
This class has the function to generate activation code and check activation code as well.

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.



```
namespace Keygen
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    2 references
    public partial class MainWindow : Window
    {
        static string privKey = "MHcwIwYKKoZIhvcNAQwBAzAVBBCB6902J6DJSmzBrqjfahPmAgEKBFDZVmOS5l+KQJ/0sC594cvf3AKkB98mBsVnrB8keYQ+Abx16IEklY1pMNiV5AWfJuEFh

        static string privRSA2048Key =
            "<?xml version=\"1.0\" encoding=\"utf-16\"?>"+
            "<RSAParameters xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">"+
              "<Exponent>AQAB</Exponent>"+
              "<Modulus>uVirzS7vy49V8EjBGBy1+zL9CD7+fjaH1Xn3COwtOUi1x5HJ1UJGlfVLafJUNPQAZeysq0GPgoYtv/uSzs73ypV4ndizGbDR1gi/qaq/CMEVh3YrOB0GFUD9JX/J3TA+Tv
              "<P>yKew0qhSbWmZcR6/4PZblHctpAP/HofrY47ZMrz5vXvFDfojziFVKcz/9Uh4LkBCsgTVNAEyPInJf+XA87CAWRM4Dc3nQkFuwMz/ggsZqn+U40dM4DZmyhk6WbQHpwOlKGUfGiZd
              "<Q>7HgH5QabtFLkBZEE17jnvdfovrEXBQd8OsMiDdYZfTjdeF0TwTk5d8sL8DdeLLpzKz53f3QaEZLzpxyX5FyEo6BLtTTSNHWq5/xq1dXlsnTzNSEPa+0d0vNfGq8YW+IUByqgkLs7
              "<DP>AuJWKlD9DlGMQRH+FA+2dUXZRBhCnH+7pGyzNoV82Xpj9LV5ukbEYJtvumjWQWPvEXv8TIkP3crlOI9m3JnW3nDmR1yETHGAVAkeSxI5xjebPkESG87sX09JhWyPB5mH0DLT5Z5
              "<DQ>XaK+BYuTVllIX4foMf/PDBaV3ATFx8EKCT7BmlcAKty8ne/FZcriHbRyWsGh8fY2qpYGvZpcxD67ap3g65VTFHPh5rQu1IUPkuMZ7ujPFtZGg/A+ag8vsxuZujhBXgjsw1W21Pi
              "<InverseQ>CUL9wy0RpsynQEC5kNkhQvHgBE9bsB5FJ3S1MNDQkljwQNEFShPyo5guVXyspLlOWMdRiT+yIP87LKa/AnFMwKIqKwvDX9jpSSxjIFUAKWrRwoU7Y1r1JQAtOShJVvfsH
              "<D>t9D88c0u/I73CArfle5DfRGgaOEwvDOQm2MOYMb6QumdfoNTzDCimG5hkH1Ijgu/qVYEGQQ4M5anINN7UadpkHyU1tbnKP932/FpQ/eW5jO4ClRjoI8ITWPaq6oEpV0HLnw/xA+b
            "</RSAParameters>";

        static string pubRSA2048Key =
            "<?xml version=\"1.0\" encoding=\"utf-16\"?>" +
            "<RSAParameters xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">" +
              "<Exponent>AQAB</Exponent>" +
              "<Modulus>uVirzS7vy49V8EjBGBy1+zL9CD7+fjaH1Xn3COwtOUi1x5HJ1UJGlfVLafJUNPQAZeysq0GPgoYtv/uSzs73ypV4ndizGbDR1gi/qaq/CMEVh3YrOB0GFUD9JX/J3TA+Tv
            "</RSAParameters>";
```



```
0 references
public static string key_generate(string devkey, string privRSA2048Key)
{
    //crypto_custom.CreateRSA2048Keys();

    byte[] dev_bt = crypto_custom.GetByteFromString(devkey);
    byte[] ret_bt = crypto_custom.merge_bytes(dev_bt, crypto_custom.CalcCRC16(dev_bt));

    UInt32 dwSeq = 0;
    byte[] bt_seq = new byte[4];
    bt_seq[0] = (byte)(dwSeq >> 24);
    bt_seq[1] = (byte)(dwSeq >> 16);
    bt_seq[2] = (byte)(dwSeq >> 8);
    bt_seq[3] = (byte)(dwSeq >> 0);

    ret_bt = crypto_custom.merge_bytes(ret_bt, bt_seq);

    UInt64 qwIssuedAt = (ulong)DateTime.UtcNow.Subtract(new DateTime(1970, 1, 1)).TotalSeconds;
    byte[] bt_issued_at = new byte[8];
    bt_issued_at[0] = (byte)(qwIssuedAt >> (24 + 32));
    bt_issued_at[1] = (byte)(qwIssuedAt >> (16 + 32));
    bt_issued_at[2] = (byte)(qwIssuedAt >> (8 + 32));
    bt_issued_at[3] = (byte)(qwIssuedAt >> (0 + 32));
    bt_issued_at[4] = (byte)(qwIssuedAt >> (24));
    bt_issued_at[5] = (byte)(qwIssuedAt >> (16));
    bt_issued_at[6] = (byte)(qwIssuedAt >> (8));
    bt_issued_at[7] = (byte)(qwIssuedAt >> (0));
```

- *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

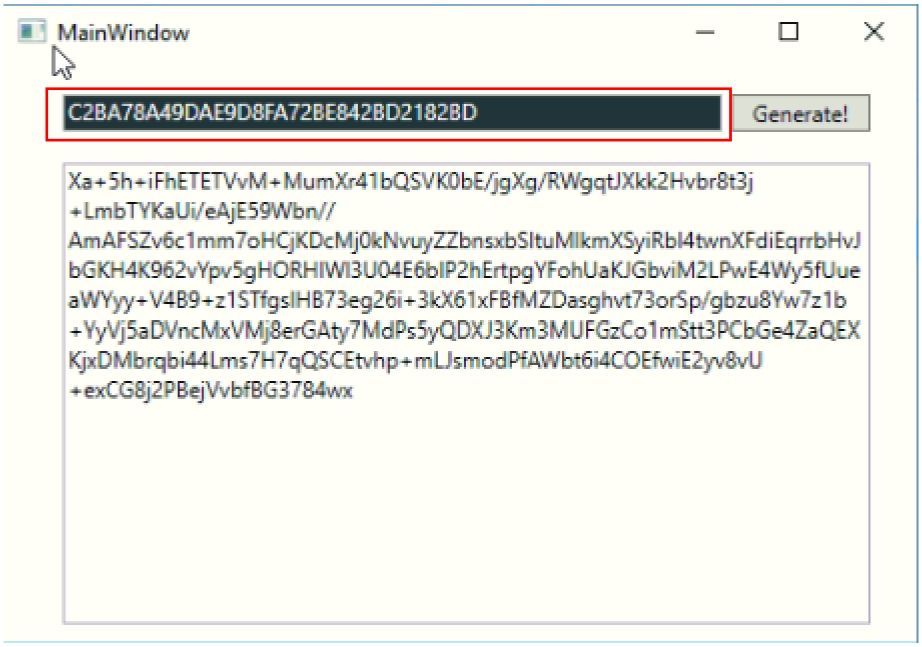
It is composed of the following data sequence.

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.

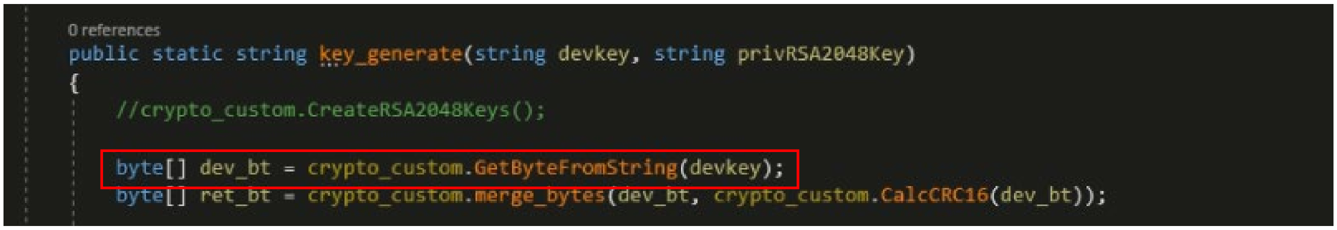
Device information is described in previous section. Provider inputs device id string in the text box of Keygen.exe



C2BA78A49DAE9D8FA72BE842BD2182BD

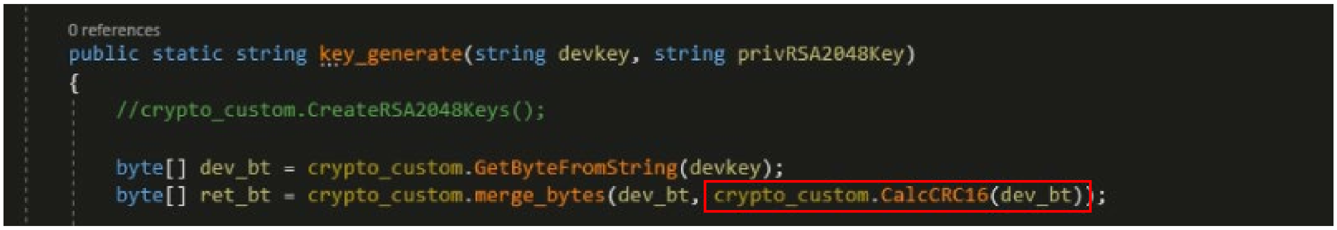
```
Xa+5h+iFhETETVvM+MumXr41bQSVK0bE/jgXg/RWgqtJXkk2Hvbr8t3j
+LmbTYKaUi/eAjE59Wbn//
AmAFSZv6c1mm7oHCjKDcMj0kNvuyZZbnsxbSltuMlkmXSyiRbl4twnXFdiEqrrbHvJ
bGKH4K962vYpv5gHORHIWI3U04E6bIP2hErtpgYFohUaKJGbviM2LPwE4Wy5fUue
aWYyy+V4B9+z1STfgsIHB73eg26i+3kX61xFBfMZDasghvt73orSp/gbzu8Yw7z1b
+YyVj5aDVncMxVMj8erGAty7MdPs5yQDXJ3Km3MUFGzCo1mStt3PCbGe4ZaQEX
KjxDMbrqbi44Lms7H7qQSCEtvhp+mLJsmodPfAWbt6i4COEfwiE2yv8vU
+exCG8j2PBejVvbfBG3784wx
```

It is MD5 string so that 32 bytes.



```
0 references
public static string key_generate(string devkey, string privRSA2048Key)
{
    //crypto_custom.CreateRSA2048Keys();

    byte[] dev_bt = crypto_custom.GetByteFromString(devkey);
    byte[] ret_bt = crypto_custom.merge_bytes(dev_bt, crypto_custom.CalcCRC16(dev_bt));
```

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

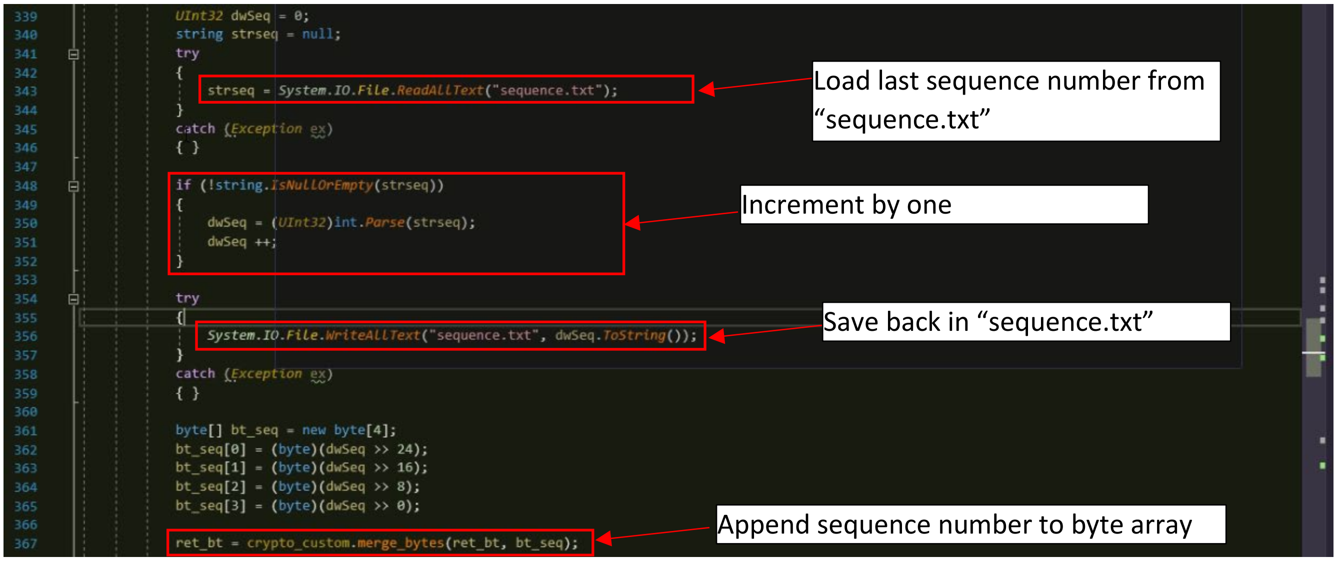


```
0 references
public static string key_generate(string devkey, string privRSA2048Key)
{
    //crypto_custom.CreateRSA2048Keys();

    byte[] dev_bt = crypto_custom.GetByteFromString(devkey);
    byte[] ret_bt = crypto_custom.merge_bytes(dev_bt, crypto_custom.CalcCRC16(dev_bt));
```

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.

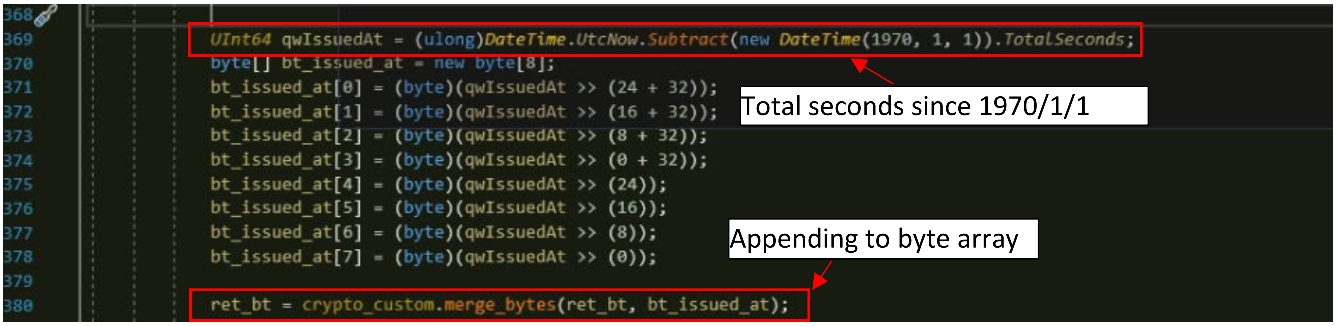
Whenever you make activation code, keygen.exe read last sequence number from “sequence.txt” and increment by one, saves it back in “sequence.txt” file.



```
UInt32 dwSeq = 0;
string strseq = null;
try
{
    strseq = System.IO.File.ReadAllText("sequence.txt");
}
catch (Exception ex)
{ }

if (!string.IsNullOrEmpty(strseq))
{
    dwSeq = (UInt32)int.Parse(strseq);
    dwSeq ++;
}

try
{
    System.IO.File.WriteAllText("sequence.txt", dwSeq.ToString());
}
catch (Exception ex)
{ }

byte[] bt_seq = new byte[4];
bt_seq[0] = (byte)(dwSeq >> 24);
bt_seq[1] = (byte)(dwSeq >> 16);
bt_seq[2] = (byte)(dwSeq >> 8);
bt_seq[3] = (byte)(dwSeq >> 0);

ret_bt = crypto_custom.merge_bytes(ret_bt, bt_seq);
```

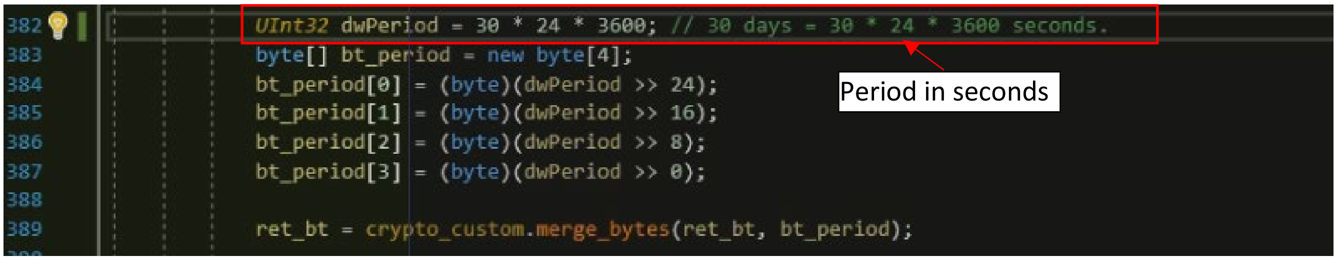
Load last sequence number from “sequence.txt”
Increment by one
Save back in “sequence.txt”
Append sequence number to byte array

“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.



```
UInt64 qwIssuedAt = (ulong)DateTime.UtcNow.Subtract(new DateTime(1970, 1, 1)).TotalSeconds;
byte[] bt_issued_at = new byte[8];
bt_issued_at[0] = (byte)(qwIssuedAt >> (24 + 32));
bt_issued_at[1] = (byte)(qwIssuedAt >> (16 + 32));
bt_issued_at[2] = (byte)(qwIssuedAt >> (8 + 32));
bt_issued_at[3] = (byte)(qwIssuedAt >> (0 + 32));
bt_issued_at[4] = (byte)(qwIssuedAt >> (24));
bt_issued_at[5] = (byte)(qwIssuedAt >> (16));
bt_issued_at[6] = (byte)(qwIssuedAt >> (8));
bt_issued_at[7] = (byte)(qwIssuedAt >> (0));

ret_bt = crypto_custom.merge_bytes(ret_bt, bt_issued_at);
```

“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.

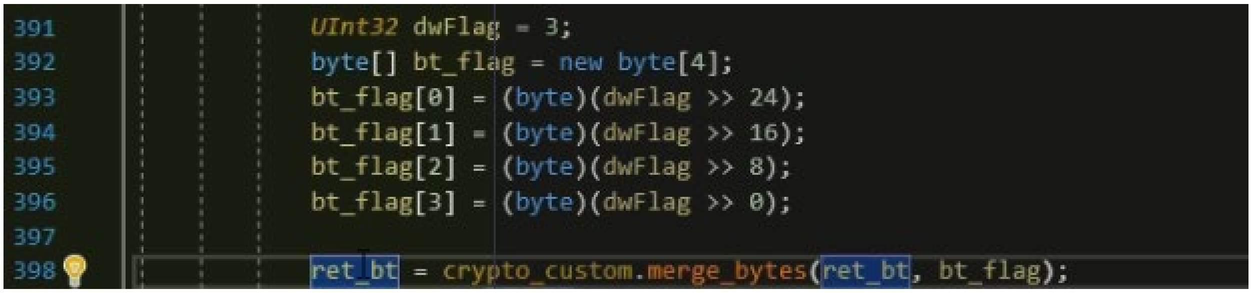


```
UInt32 dwPeriod = 30 * 24 * 3600; // 30 days = 30 * 24 * 3600 seconds.
byte[] bt_period = new byte[4];
bt_period[0] = (byte)(dwPeriod >> 24);
bt_period[1] = (byte)(dwPeriod >> 16);
bt_period[2] = (byte)(dwPeriod >> 8);
bt_period[3] = (byte)(dwPeriod >> 0);

ret_bt = crypto_custom.merge_bytes(ret_bt, bt_period);
```

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

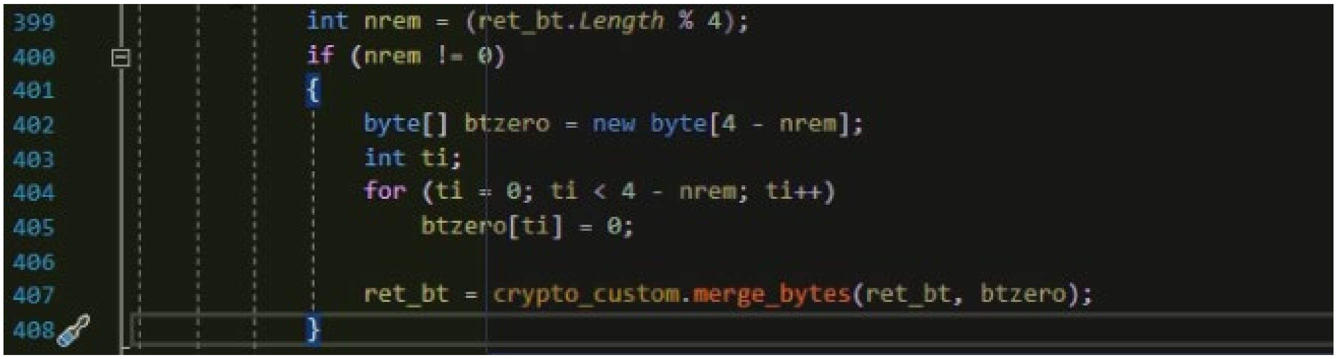
```
dwPeriod = 365 * 24 * 3600;
```

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



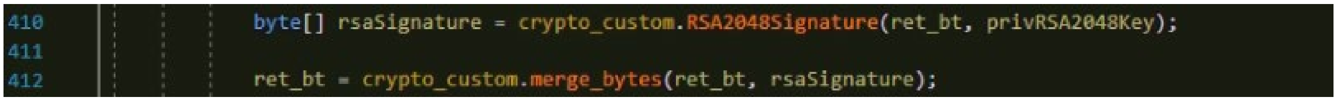
```
            UInt32 dwFlag = 3;
            byte[] bt_flag = new byte[4];
            bt_flag[0] = (byte)(dwFlag >> 24);
            bt_flag[1] = (byte)(dwFlag >> 16);
            bt_flag[2] = (byte)(dwFlag >> 8);
            bt_flag[3] = (byte)(dwFlag >> 0);

            ret_bt = crypto_custom.merge_bytes(ret_bt, bt_flag);
```

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.



```
            int nrem = (ret_bt.Length % 4);
            if (nrem != 0)
            {
                byte[] btzero = new byte[4 - nrem];
                int ti;
                for (ti = 0; ti < 4 - nrem; ti++)
                    btzero[ti] = 0;

                ret_bt = crypto_custom.merge_bytes(ret_bt, btzero);
            }
```

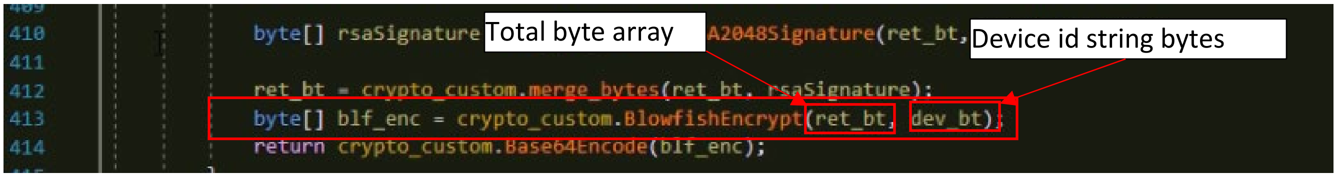
We calculate RSA2048 signature of the byte array.



```
            byte[] rsaSignature = crypto_custom.RSA2048Signature(ret_bt, privRSA2048Key);

            ret_bt = crypto_custom.merge_bytes(ret_bt, rsaSignature);
```

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.



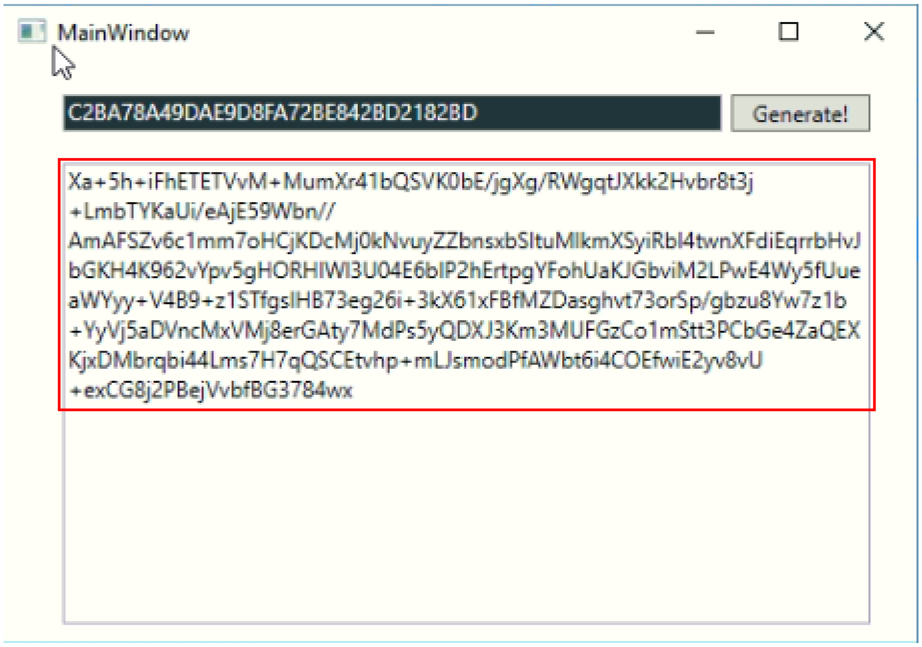
```
            byte[] rsaSignature                  A2048Signature(ret_bt,

            ret_bt = crypto_custom.merge_bytes(ret_bt, rsaSignature);
            byte[] blf_enc = crypto_custom.BlowfishEncrypt(ret_bt, dev_bt);
            return crypto_custom.Base64Encode(blf_enc);
```

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.



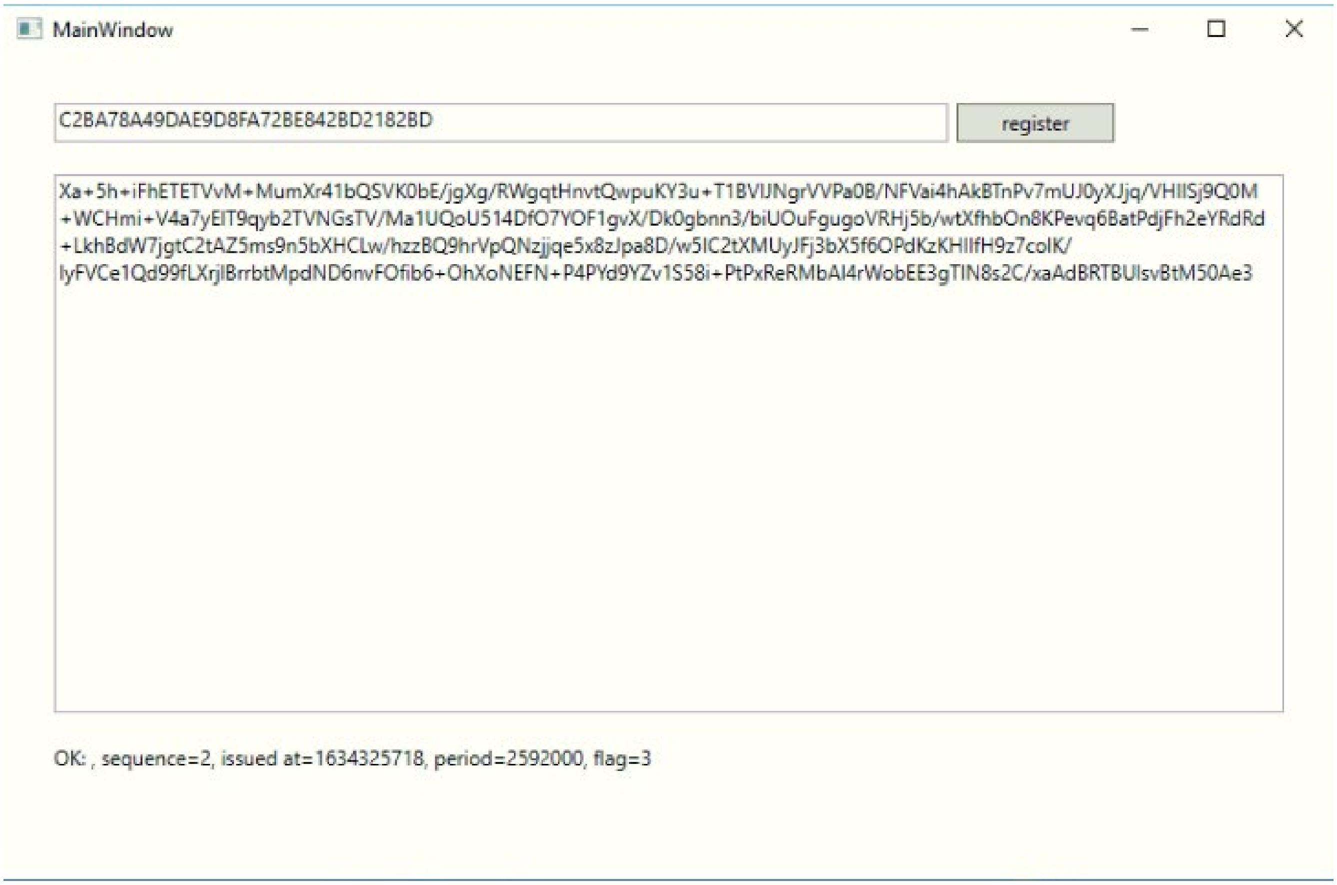
```
C2BA78A49DAE9D8FA72BE842BD2182BD
```

```
Xa+5h+iFhETETVvM+MumXr41bQSVK0bE/jgXg/RWgqtJXkk2Hvbr8t3j
+LmbTYKaUi/eAjE59Wbn//
AmAFSZv6c1mm7oHCjKDcMj0kNvuyZZbnsxbSltuMlkmXSyiRbl4twnXFdiEqrrbHvJ
bGKH4K962vYpv5gHORHIWI3U04E6bIP2hErtpgYFohUaKJGbviM2LPwE4Wy5fUue
aWYyy+V4B9+z1STfgsIHB73eg26i+3kX61xFBfMZDasghvt73orSp/gbzu8Yw7z1b
+YyVj5aDVncMxVMj8erGAty7MdPs5yQDXJ3Km3MUFGzCo1mStt3PCbGe4ZaQEX
KjxDMbrqbi44Lms7H7qQSCEtvhp+mLJsmodPfAWbt6i4COEfwiE2yv8vU
+exCG8j2PBejVvbfBG3784wx
```

2) 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”.



```
C2BA78A49DAE9D8FA72BE842BD2182BD
```

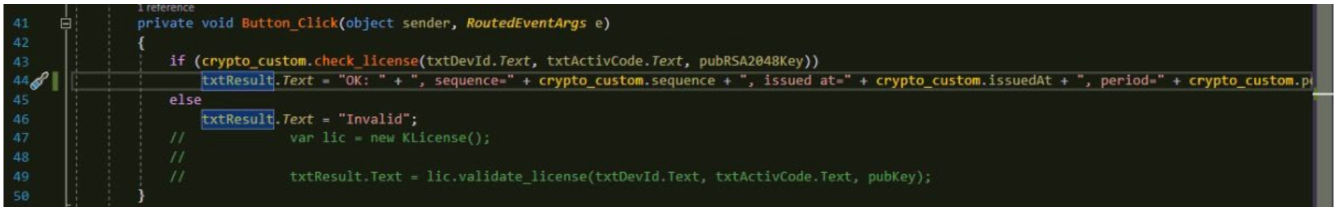
```
Xa+5h+iFhETETVvM+MumXr41bQSVK0bE/jgXg/RWgqtHnvtQwpuKY3u+T1BVIJNgrVVPa0B/NFVai4hAkBTnPv7mUJ0yXJjq/VHIISj9Q0M
+WCHmi+V4a7yElT9qyb2TVNGsTV/Ma1UQoU514DfO7YOF1gvX/Dk0gbnn3/biUOuFgugoVRHj5b/wtXfhbOn8KPevq6BatPdjFh2eYRdRd
+LkhBdW7jgtC2tAZ5ms9n5bXHCLw/hzzBQ9hrVpQNzjjqe5x8zJpa8D/w5IC2tXMUyJFj3bX5f6OPdKzKHIIfH9z7colK/
IyFVCe1Qd99fLXrjIBrrbtMpdND6nvFOfib6+OhXoNEFN+P4PYd9YZv1S58i+PtPxReRMbAI4rWobEE3gTIN8s2C/xaAdBRTBUIsvBtM50Ae3
```

```
OK: , sequence=2, issued at=1634325718, period=2592000, flag=3
```

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.



```
private void Button_Click(object sender, RoutedEventArgs e)
{
    if (crypto_custom.check_license(txtDevId.Text, txtActivCode.Text, pubRSA2048Key))
        txtResult.Text = "OK: " + ", sequence=" + crypto_custom.sequence + ", issued at=" + crypto_custom.issuedAt + ", period=" + crypto_custom.pe
    else
        txtResult.Text = "Invalid";
    //              var lic = new KLicense();
    //
    //              txtResult.Text = lic.validate_license(txtDevId.Text, txtActivCode.Text, pubKey);
}
```

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.

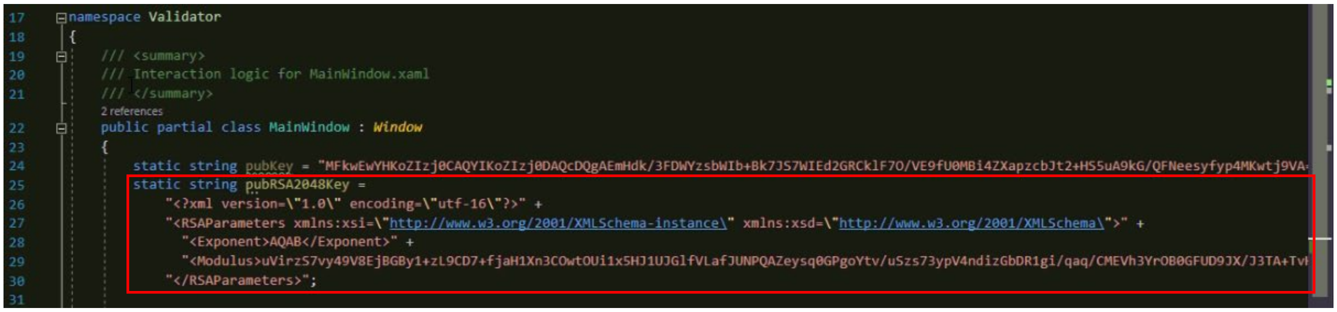
In this function, check_license() of the class “crypto_custom” is used. This returns “true” or “false” which means the activation code is valid or not.

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.

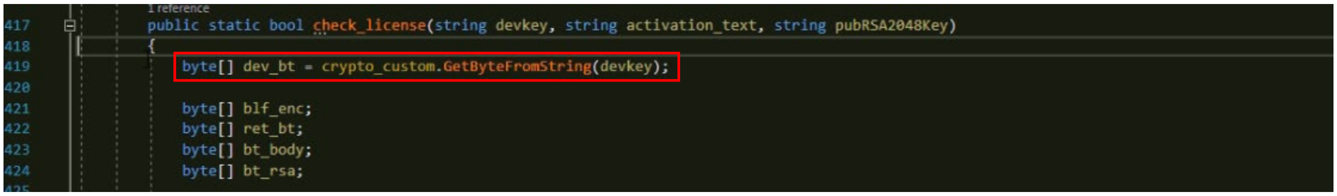
RSA2048 public key is stored in “Validator” project.



```
namespace Validator
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        static string pubKey = "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEmHdk/3FDWYzsbWIb+Bk7JS7WIEd2GRCklF7O/VE9fU0MBi4ZXapzcbJt2+HS5uA9kG/QFNeesyfyp4MKwtj9VA
        static string pubRSA2048Key =
            "<?xml version=\"1.0\" encoding=\"utf-16\"?>" +
            "<RSAParameters xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xmlns:xsd=\"http://www.w3.org/2001/XMLSchema\">" +
              "<Exponent>AQAB</Exponent>" +
              "<Modulus>uVirzS7vy49V8EjBGBy1+zL9CD7+fjaH1Xn3COwtOUi1x5HJ1UJGlfVLafJUNPQAZeysq0GPgoYtv/uSzs73ypV4ndizGbDR1gi/qaq/CMEVh3YrOB0GFUD9JX/J3TA+Tv
            "</RSAParameters>";
```

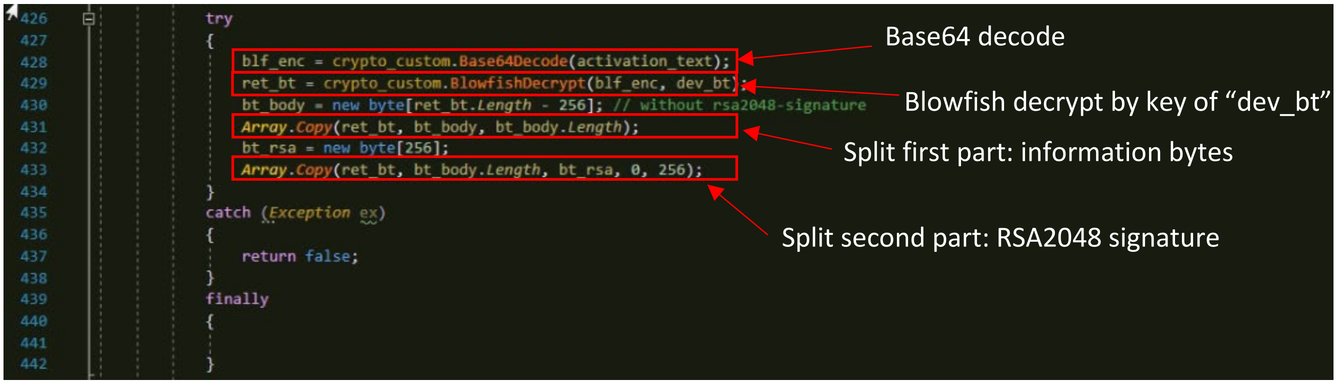
This public key should be paired to the private key in “Keygen” project.

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



```
public static bool check_license(string devkey, string activation_text, string pubRSA2048Key)
{
    byte[] dev_bt = crypto_custom.GetByteFromString(devkey);

    byte[] blf_enc;
    byte[] ret_bt;
    byte[] bt_body;
    byte[] bt_rsa;
```

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



```
try
{
    blf_enc = crypto_custom.Base64Decode(activation_text);
    ret_bt = crypto_custom.BlowfishDecrypt(blf_enc, dev_bt);
    bt_body = new byte[ret_bt.Length - 256]; // without rsa2048-signature
    Array.Copy(ret_bt, bt_body, bt_body.Length);
    bt_rsa = new byte[256];
    Array.Copy(ret_bt, bt_body.Length, bt_rsa, 0, 256);
}
catch (Exception ex)
{
    return false;
}
finally
{

}
```

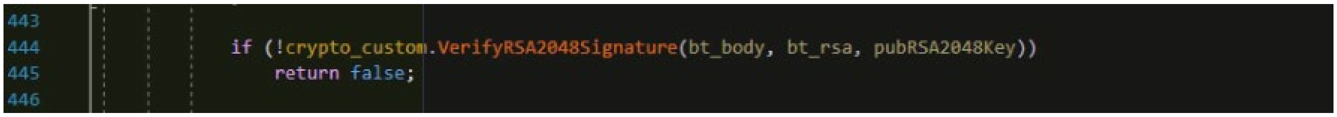
Base64 decode

Blowfish decrypt by key of “dev_bt”

Split first part: information bytes

Split second part: RSA2048 signature

We check validity of RSA2048 signature.

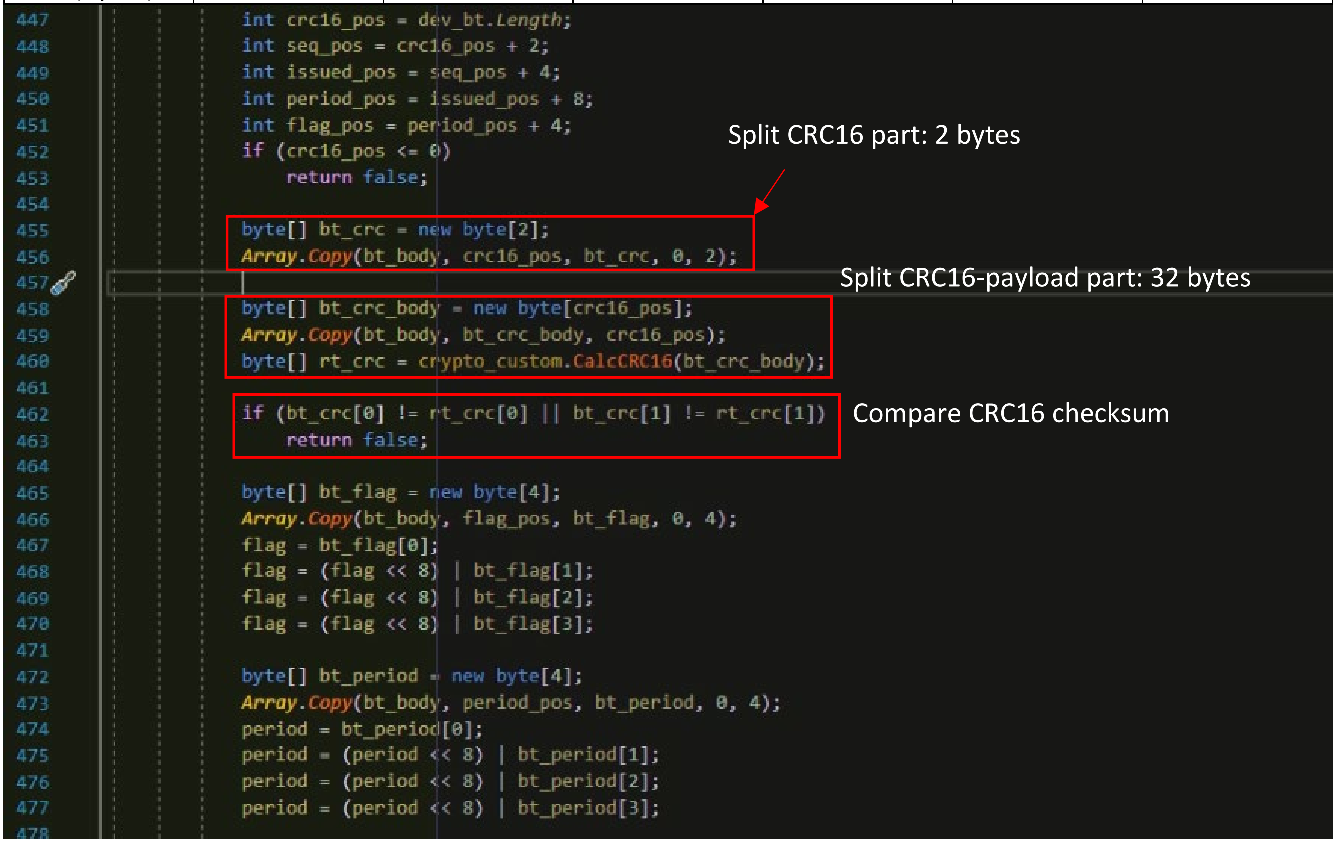


```
if (!crypto_custom.VerifyRSA2048Signature(bt_body, bt_rsa, pubRSA2048Key))
    return false;
```

- 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

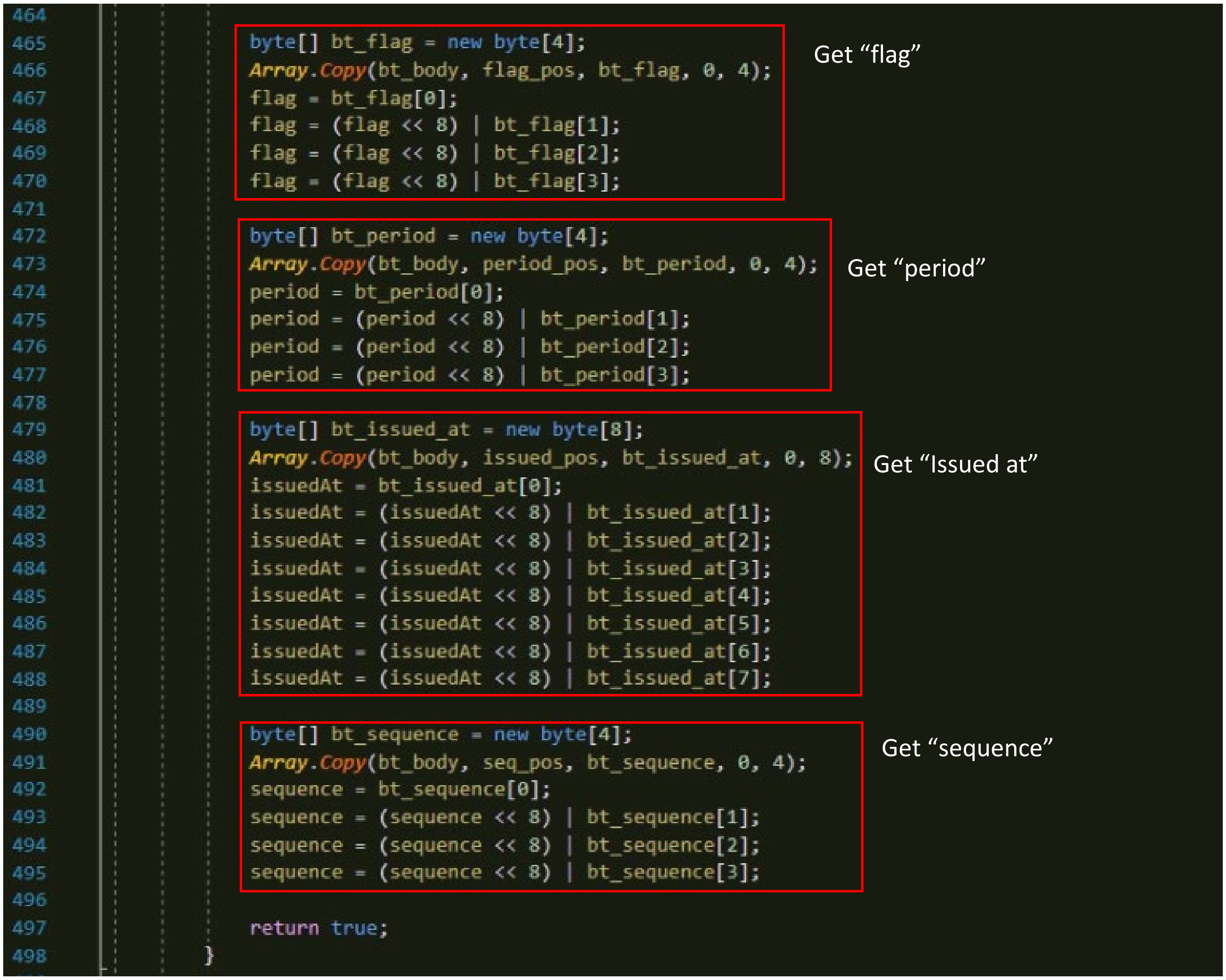


```
int crc16_pos = dev_bt.Length;
int seq_pos = crc16_pos + 2;
int issued_pos = seq_pos + 4;
int period_pos = issued_pos + 8;
int flag_pos = period_pos + 4;
if (crc16_pos <= 0)
    return false;

byte[] bt_crc = new byte[2];
Array.Copy(bt_body, crc16_pos, bt_crc, 0, 2);

byte[] bt_crc_body = new byte[crc16_pos];
Array.Copy(bt_body, bt_crc_body, crc16_pos);
byte[] rt_crc = crypto_custom.CalcCRC16(bt_crc_body);

if (bt_crc[0] != rt_crc[0] || bt_crc[1] != rt_crc[1])
    return false;

byte[] bt_flag = new byte[4];
Array.Copy(bt_body, flag_pos, bt_flag, 0, 4);
flag = bt_flag[0];
flag = (flag << 8) | bt_flag[1];
flag = (flag << 8) | bt_flag[2];
flag = (flag << 8) | bt_flag[3];

byte[] bt_period = new byte[4];
Array.Copy(bt_body, period_pos, bt_period, 0, 4);
period = bt_period[0];
period = (period << 8) | bt_period[1];
period = (period << 8) | bt_period[2];
period = (period << 8) | bt_period[3];
```

Split CRC16 part: 2 bytes

Split CRC16-payload part: 32 bytes

Compare CRC16 checksum



```
        byte[] bt_flag = new byte[4];
        Array.Copy(bt_body, flag_pos, bt_flag, 0, 4);
        flag = bt_flag[0];
        flag = (flag << 8) | bt_flag[1];
        flag = (flag << 8) | bt_flag[2];
        flag = (flag << 8) | bt_flag[3];

        byte[] bt_period = new byte[4];
        Array.Copy(bt_body, period_pos, bt_period, 0, 4);
        period = bt_period[0];
        period = (period << 8) | bt_period[1];
        period = (period << 8) | bt_period[2];
        period = (period << 8) | bt_period[3];

        byte[] bt_issued_at = new byte[8];
        Array.Copy(bt_body, issued_pos, bt_issued_at, 0, 8);
        issuedAt = bt_issued_at[0];
        issuedAt = (issuedAt << 8) | bt_issued_at[1];
        issuedAt = (issuedAt << 8) | bt_issued_at[2];
        issuedAt = (issuedAt << 8) | bt_issued_at[3];
        issuedAt = (issuedAt << 8) | bt_issued_at[4];
        issuedAt = (issuedAt << 8) | bt_issued_at[5];
        issuedAt = (issuedAt << 8) | bt_issued_at[6];
        issuedAt = (issuedAt << 8) | bt_issued_at[7];

        byte[] bt_sequence = new byte[4];
        Array.Copy(bt_body, seq_pos, bt_sequence, 0, 4);
        sequence = bt_sequence[0];
        sequence = (sequence << 8) | bt_sequence[1];
        sequence = (sequence << 8) | bt_sequence[2];
        sequence = (sequence << 8) | bt_sequence[3];

        return true;
    }
```

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

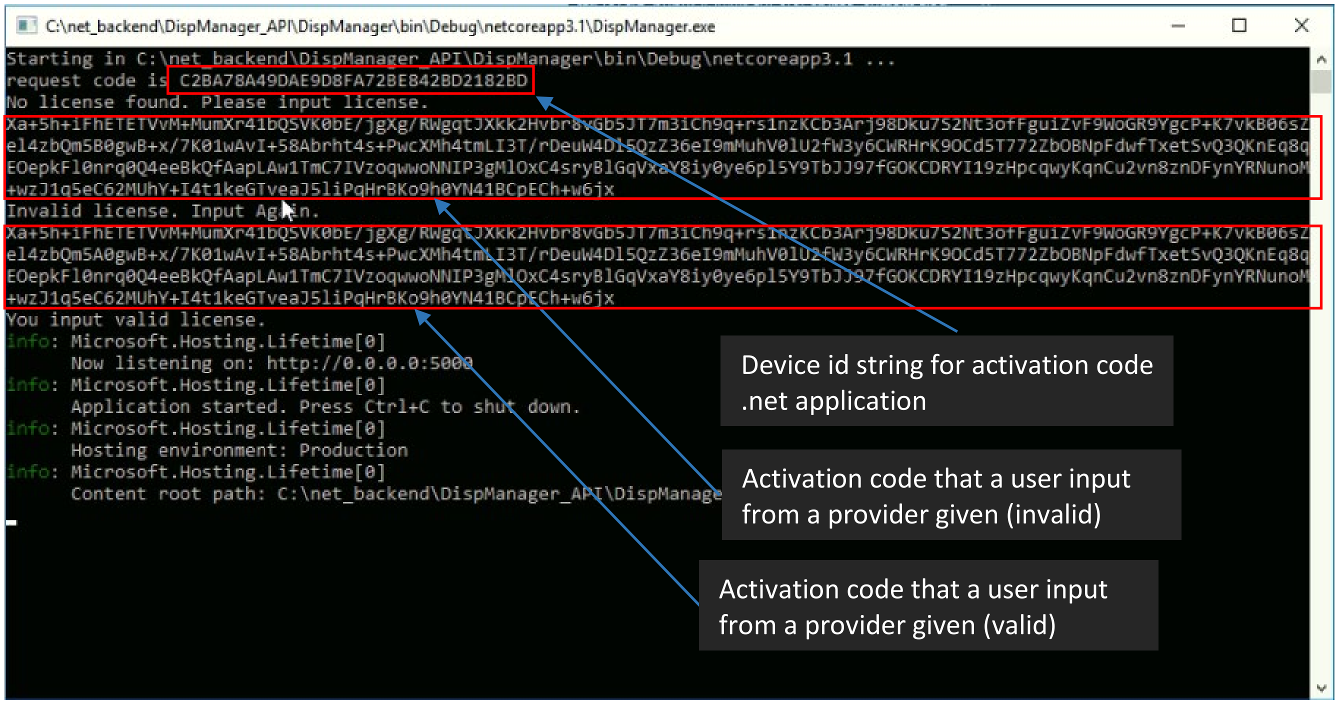
UInt32	crypto_custom.sequence

3. Licensing system in .net application

1) 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.



C:\net_backend\DispManager_API\DispManager\bin\Debug\netcoreapp3.1\DispManager.exe

```
Starting in C:\net_backend\DispManager_API\DispManager\bin\Debug\netcoreapp3.1 ...
request code is C2BA78A49DAE9D8FA72BE842BD2182BD
No license found. Please input license.
Xa+5h+iFhETETVvM+MumXr41bQSVK0bE/jgXg/RWgqtJXkk2Hvbr8vGb5JT7m3iCh9q+rs1nzKCb3Arj98Dku7S2Nt3ofFguiZvF9WoGR9YgcP+K7vkB06sZ
el4zbQm5B0gwB+x/7K01wAvI+58Abrht4s+PwcXMh4tmLI3T/rDeuW4Dl5QzZ36eI9mMuhV0lU2fW3y6CWRHrK9OCd5T772ZbOBNpFdwfTxetSvQ3QKnEq8q
EOepkFl0nrq0Q4eeBkQfAapLAw1TmC7IVzoqwwoNNIP3gMlOxC4sryBlGqVxaY8iy0ye6pl5Y9TbJJ97fGOKCDRYI19zHpcqwyKqnCu2vn8znDFynYRNunoM
+wzJ1q5eC62MUhY+I4t1keGTveaJ5liPqHrBKo9h0YN41BCpECh+w6jx
Invalid license. Input Again.
Xa+5h+iFhETETVvM+MumXr41bQSVK0bE/jgXg/RWgqtJXkk2Hvbr8vGb5JT7m3iCh9q+rs1nzKCb3Arj98Dku7S2Nt3ofFguiZvF9WoGR9YgcP+K7vkB06sZ
el4zbQm5A0gwB+x/7K01wAvI+58Abrht4s+PwcXMh4tmLI3T/rDeuW4Dl5QzZ36eI9mMuhV0lU2fW3y6CWRHrK9OCd5T772ZbOBNpFdwfTxetSvQ3QKnEq8q
EOepkFl0nrq0Q4eeBkQfAapLAw1TmC7IVzoqwwoNNIP3gMlOxC4sryBlGqVxaY8iy0ye6pl5Y9TbJJ97fGOKCDRYI19zHpcqwyKqnCu2vn8znDFynYRNunoM
+wzJ1q5eC62MUhY+I4t1keGTveaJ5liPqHrBKo9h0YN41BCpECh+w6jx
You input valid license.
info: Microsoft.Hosting.Lifetime[0]
      Now listening on: http://0.0.0.0:5000
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Production
info: Microsoft.Hosting.Lifetime[0]
      Content root path: C:\net_backend\DispManager_API\DispManage
```

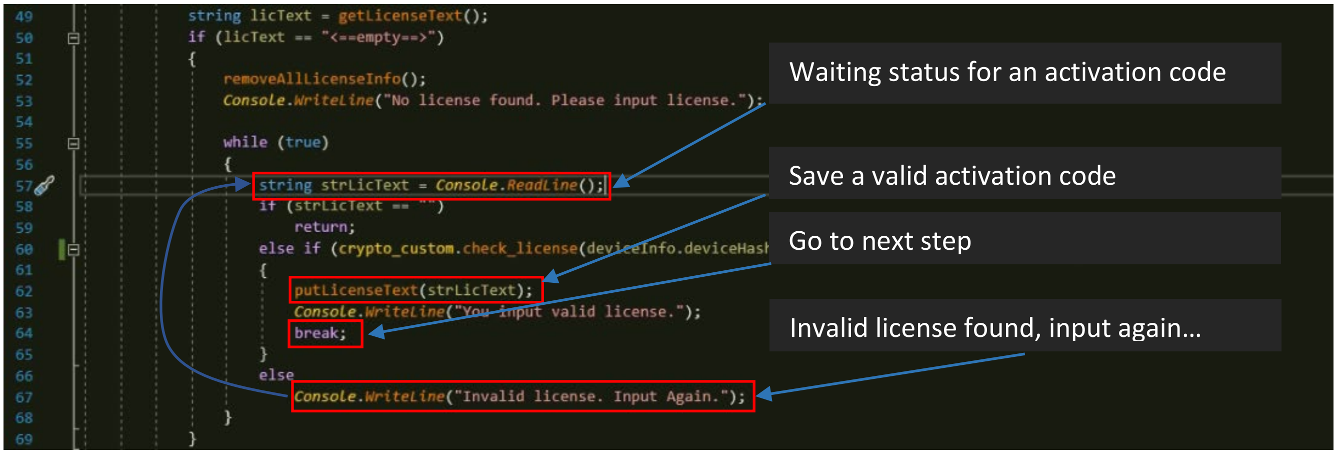
Device id string for activation code .net application

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

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

.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.



```
            string licText = getLicenseText();
            if (licText == "<==empty==>")
            {
                removeAllLicenseInfo();
                Console.WriteLine("No license found. Please input license.");

                while (true)
                {
                    string strLicText = Console.ReadLine();
                    if (strLicText == "")
                        return;
                    else if (crypto_custom.check_license(deviceInfo.deviceHash
                    {
                        putLicenseText(strLicText);
                        Console.WriteLine("You input valid license.");
                        break;
                    }
                    else
                        Console.WriteLine("Invalid license. Input Again.");
                }
            }
```

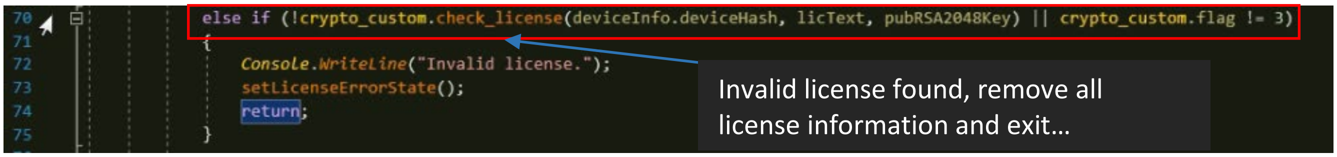
Waiting status for an activation code

Save a valid activation code

Go to next step

Invalid license found, input again…

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



```
            else if (!crypto_custom.check_license(deviceInfo.deviceHash, licText, pubRSA2048Key) || crypto_custom.flag != 3)
            {
                Console.WriteLine("Invalid license.");
                setLicenseErrorState();
                return;
            }
```

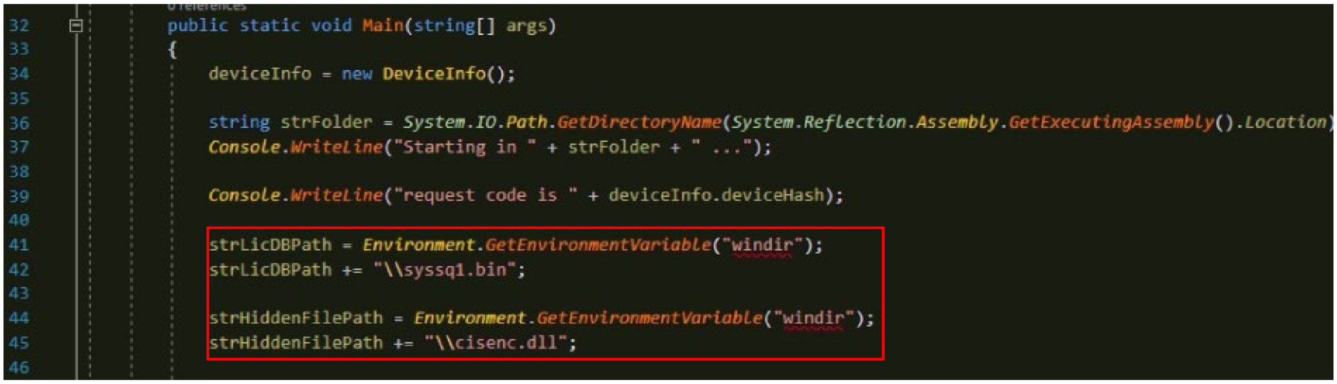
Invalid license found, remove all license information and exit…

- *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.

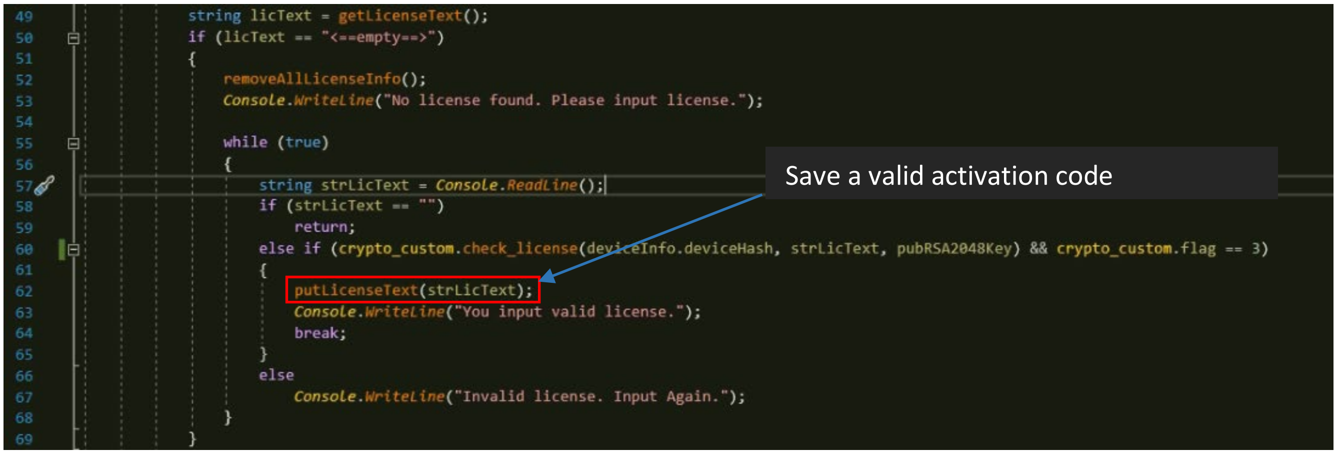


```
        0 references
        public static void Main(string[] args)
        {
            deviceInfo = new DeviceInfo();

            string strFolder = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location)
            Console.WriteLine("Starting in " + strFolder + " ...");

            Console.WriteLine("request code is " + deviceInfo.deviceHash);

            strLicDBPath = Environment.GetEnvironmentVariable("windir");
            strLicDBPath += "\\syssq1.bin";

            strHiddenFilePath = Environment.GetEnvironmentVariable("windir");
            strHiddenFilePath += "\\cisenc.dll";

```

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.



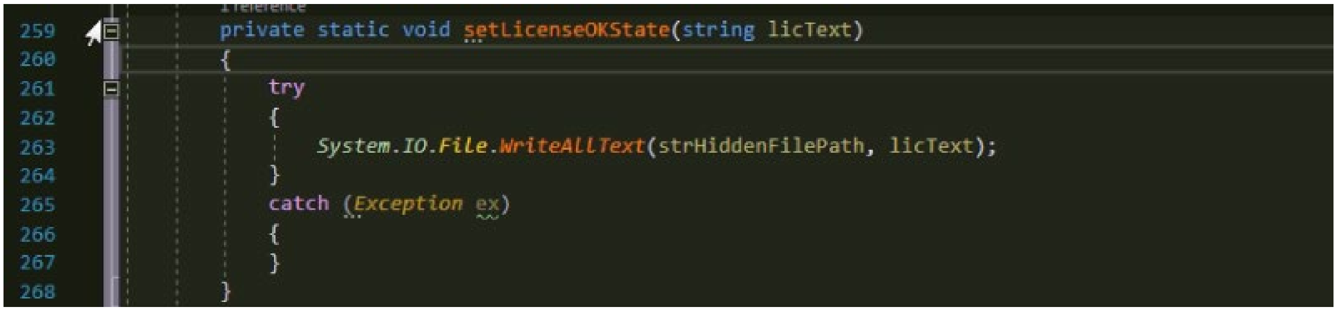
```
            string licText = getLicenseText();
            if (licText == "<==empty==>")
            {
                removeAllLicenseInfo();
                Console.WriteLine("No license found. Please input license.");

                while (true)
                {
                    string strLicText = Console.ReadLine();
                    if (strLicText == "")
                        return;
                    else if (crypto_custom.check_license(deviceInfo.deviceHash, strLicText, pubRSA2048Key) && crypto_custom.flag == 3)
                    {
                        putLicenseText(strLicText);
                        Console.WriteLine("You input valid license.");
                        break;
                    }
                    else
                        Console.WriteLine("Invalid license. Input Again.");
                }
            }
```

Save a valid activation code

This function creates encrypted sqlite db file and hidden file.

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

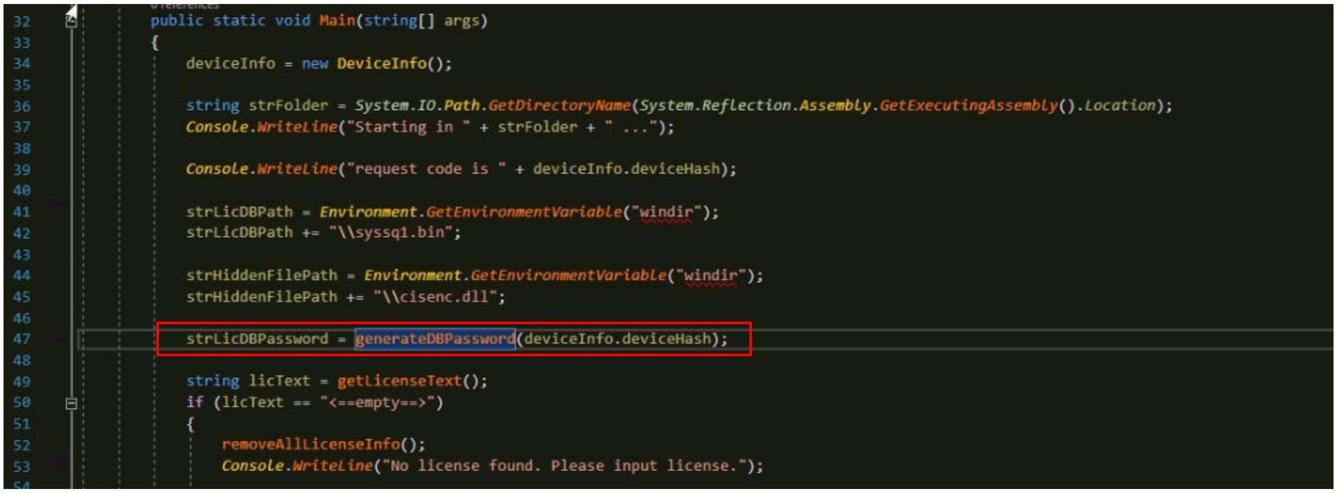
It writes the whole part of activation code to a hidden file.



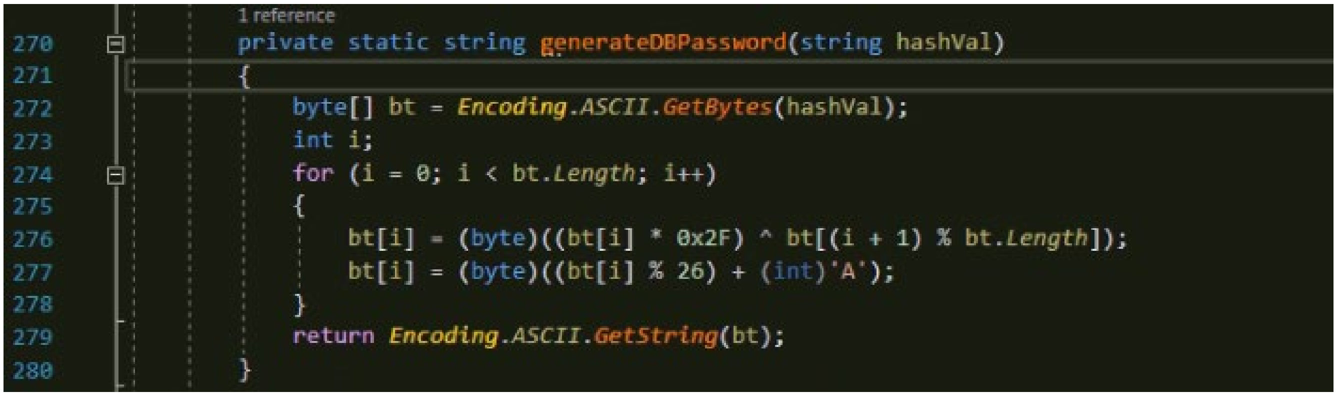
```
        1 reference
        private static void setLicenseOKState(string licText)
        {
            try
            {
                System.IO.File.WriteAllText(strHiddenFilePath, licText);
            }
            catch (Exception ex)
            {
            }
        }
```

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.

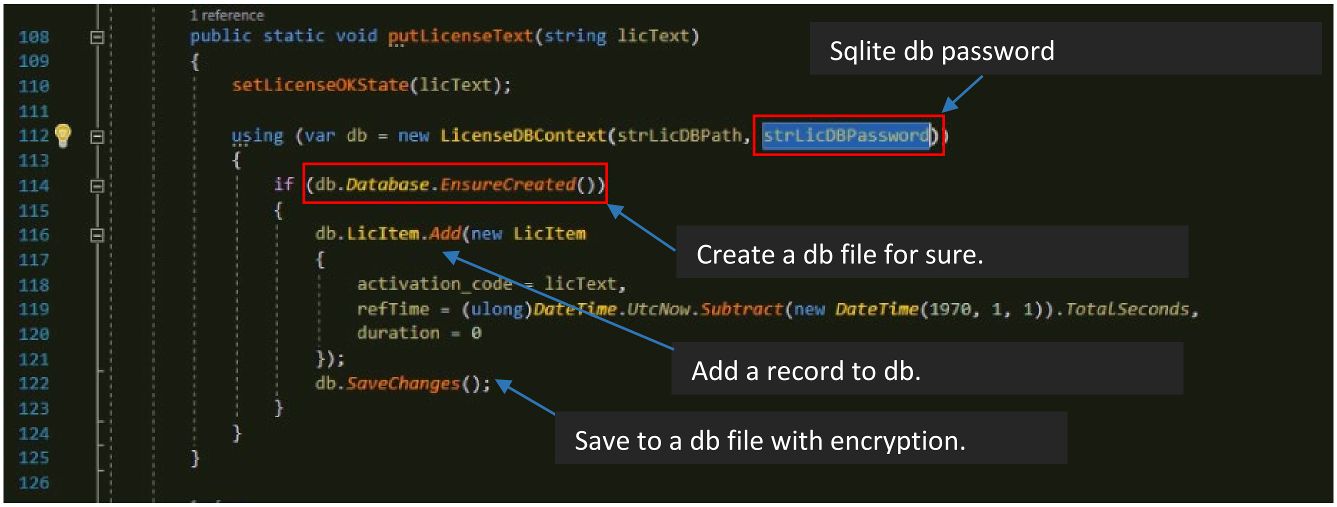


```
public static void Main(string[] args)
{
    deviceInfo = new DeviceInfo();

    string strFolder = System.IO.Path.GetDirectoryName(System.Reflection.Assembly.GetExecutingAssembly().Location);
    Console.WriteLine("Starting in " + strFolder + " ...");

    Console.WriteLine("request code is " + deviceInfo.deviceHash);

    strLicDBPath = Environment.GetEnvironmentVariable("windir");
    strLicDBPath += "\\syssq1.bin";

    strHiddenFilePath = Environment.GetEnvironmentVariable("windir");
    strHiddenFilePath += "\\cisenc.dll";

    strLicDBPassword = generateDBPassword(deviceInfo.deviceHash);

    string licText = getLicenseText();
    if (licText == "<==empty==>")
    {
        removeAllLicenseInfo();
        Console.WriteLine("No license found. Please input license.");
```



```
1 reference
private static string generateDBPassword(string hashVal)
{
    byte[] bt = Encoding.ASCII.GetBytes(hashVal);
    int i;
    for (i = 0; i < bt.Length; i++)
    {
        bt[i] = (byte)((bt[i] * 0x2F) ^ bt[(i + 1) % bt.Length]);
        bt[i] = (byte)((bt[i] % 26) + (int)'A');
    }
    return Encoding.ASCII.GetString(bt);
}
```

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



```
1 reference
public static void putLicenseText(string licText)
{
    setLicenseOKState(licText);

    using (var db = new LicenseDBContext(strLicDBPath, strLicDBPassword))
    {
        if (db.Database.EnsureCreated())
        {
            db.LicItem.Add(new LicItem
            {
                activation_code = licText,
                refTime = (ulong)DateTime.UtcNow.Subtract(new DateTime(1970, 1, 1)).TotalSeconds,
                duration = 0
            });
            db.SaveChanges();
        }
    }
}
```

Sqlite db password

Create a db file for sure.

Add a record to db.

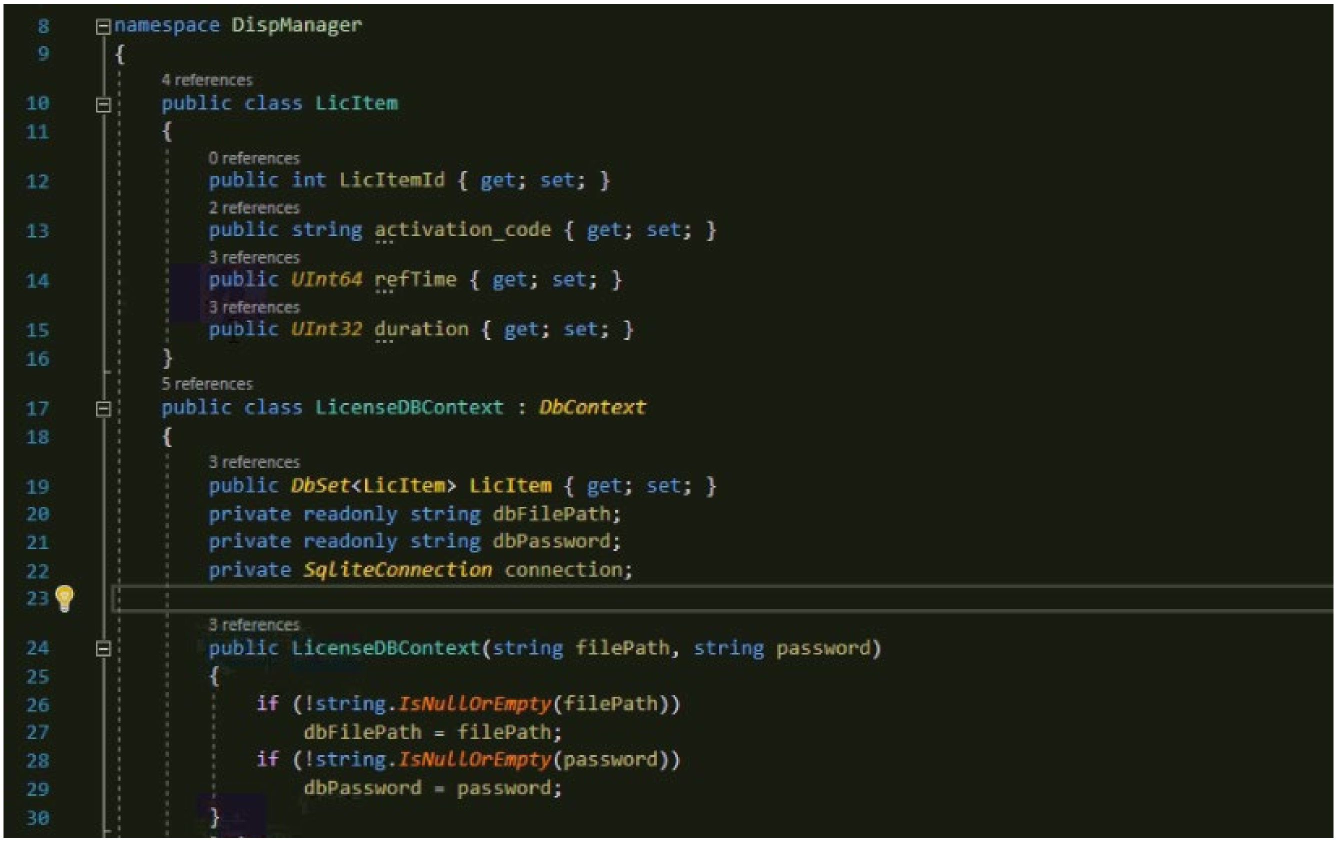
Save to a db file with encryption.

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



```
namespace DispManager
{
    4 references
    public class LicItem
    {
        0 references
        public int LicItemId { get; set; }
        2 references
        public string activation_code { get; set; }
        3 references
        public UInt64 refTime { get; set; }
        3 references
        public UInt32 duration { get; set; }
    }
    5 references
    public class LicenseDBContext : DbContext
    {
        3 references
        public DbSet<LicItem> LicItem { get; set; }
        private readonly string dbFilePath;
        private readonly string dbPassword;
        private SqliteConnection connection;

        3 references
        public LicenseDBContext(string filePath, string password)
        {
            if (!string.IsNullOrEmpty(filePath))
                dbFilePath = filePath;
            if (!string.IsNullOrEmpty(password))
                dbPassword = password;
        }
```

- *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.

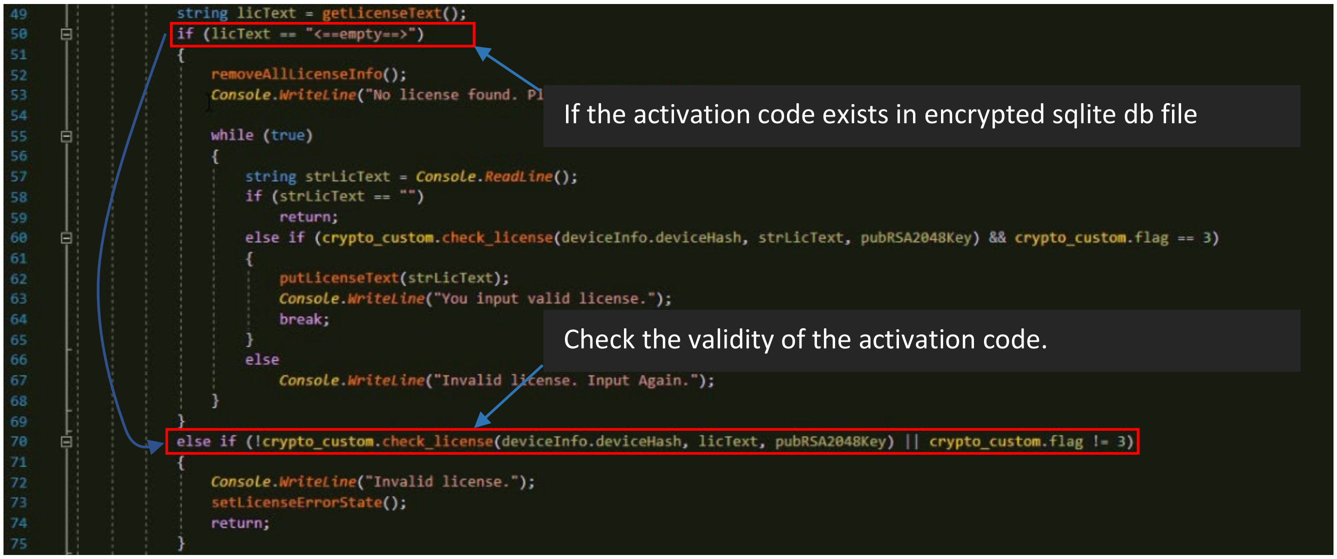
But to make sqlite3.dll file, you should know the source project that built this sqlite3.dll file. If you can get the right version of source code project, then you can add password/encryption code to the project and rebuild the project to product sqlite3.dll that supports password/encryption.

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

2) 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.



```
string licText = getLicenseText();
if (licText == "<==empty==>")
{
    removeAllLicenseInfo();
    Console.WriteLine("No license found. Pl

    while (true)
    {
        string strLicText = Console.ReadLine();
        if (strLicText == "")
            return;
        else if (crypto_custom.check_license(deviceInfo.deviceHash, strLicText, pubRSA2048Key) && crypto_custom.flag == 3)
        {
            putLicenseText(strLicText);
            Console.WriteLine("You input valid license.");
            break;
        }
        else
            Console.WriteLine("Invalid license. Input Again.");
    }
}
else if (!crypto_custom.check_license(deviceInfo.deviceHash, licText, pubRSA2048Key) || crypto_custom.flag != 3)
{
    Console.WriteLine("Invalid license.");
    setLicenseErrorState();
    return;
}
```

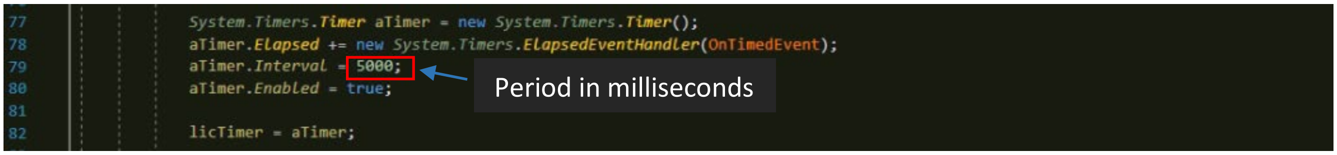
If the activation code exists in encrypted sqlite db file

Check the validity of the activation code.

3) 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.



```
System.Timers.Timer aTimer = new System.Timers.Timer();
aTimer.Elapsed += new System.Timers.ElapsedEventHandler(OnTimedEvent);
aTimer.Interval = 5000;
aTimer.Enabled = true;

licTimer = aTimer;
```

Period in milliseconds

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

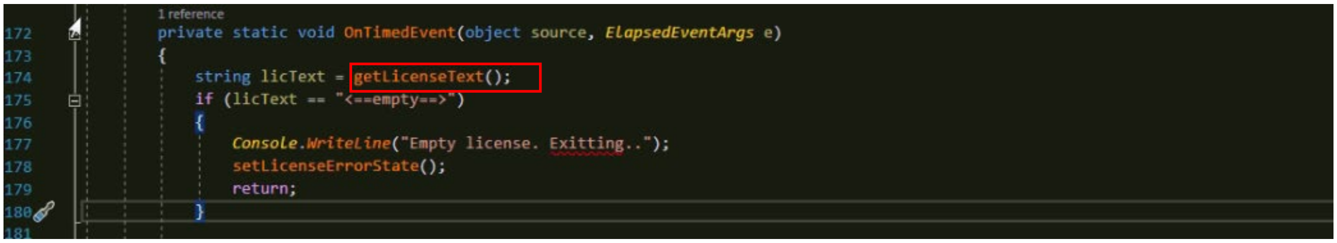
When the timer is timed out, it calls the callback function “OnTimedEvent”.

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

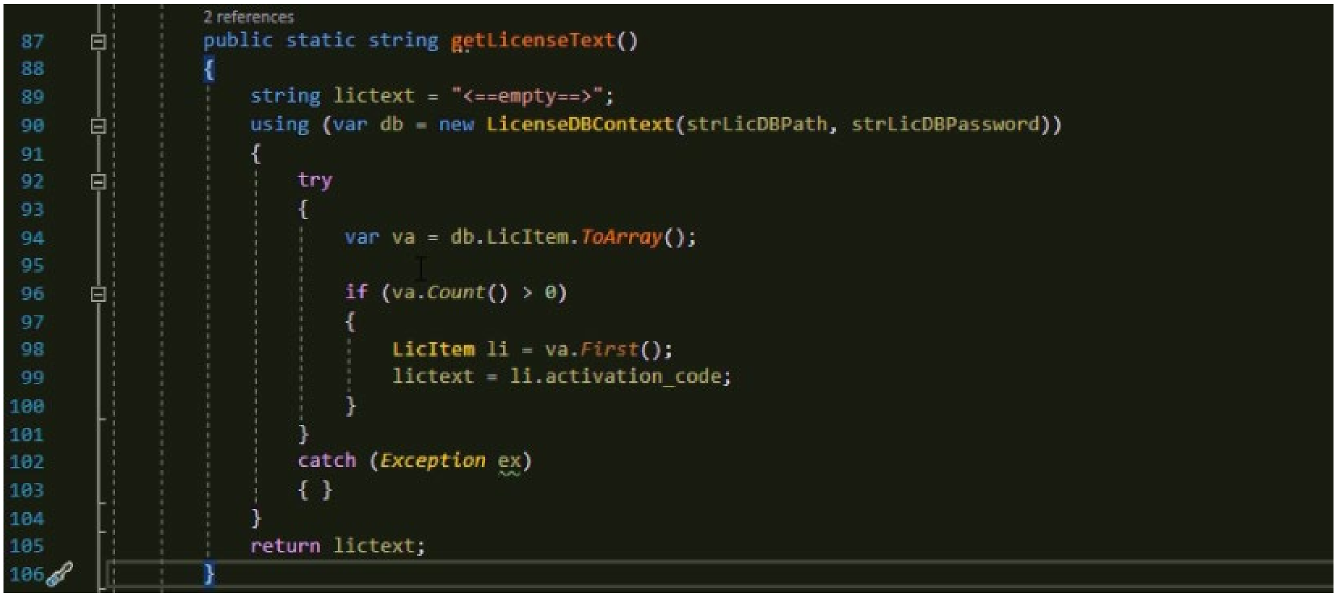
- *Get activation code from encrypted sqlite db file*

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.



```
private static void OnTimedEvent(object source, ElapsedEventArgs e)
{
    string licText = getLicenseText();
    if (licText == "<==empty==>")
    {
        Console.WriteLine("Empty license. Exitting..");
        setLicenseErrorState();
        return;
    }
```

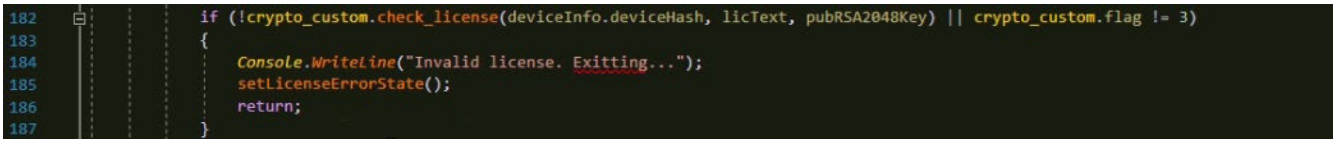
getLicenseText() function retrieves the activation code from encrypted sqlite db file that a user input.



```
public static string getLicenseText()
{
    string lictext = "<==empty==>";
    using (var db = new LicenseDBContext(strLicDBPath, strLicDBPassword))
    {
        try
        {
            var va = db.LicItem.ToArray();

            if (va.Count() > 0)
            {
                LicItem li = va.First();
                lictext = li.activation_code;
            }
        }
        catch (Exception ex)
        { }
    }
    return lictext;
}
```

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

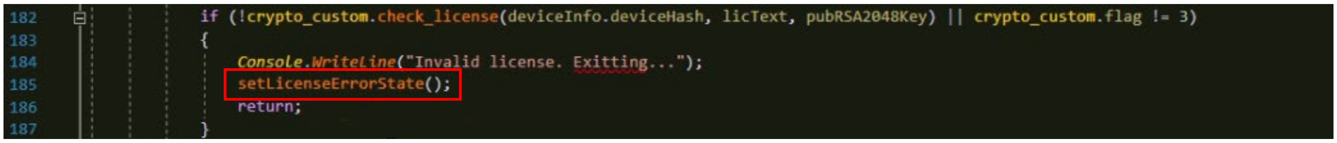
After retrieving the activation code from an encrypted sqlite db file, we check its validity.



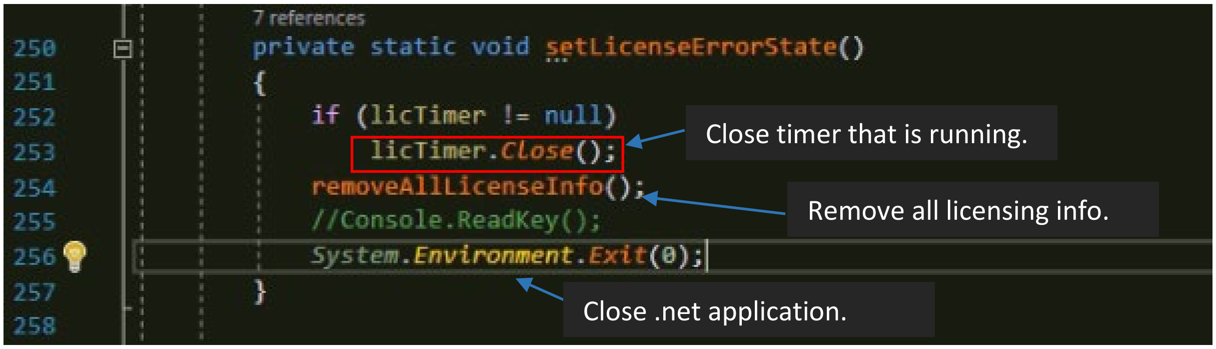
```
if (!crypto_custom.check_license(deviceInfo.deviceHash, licText, pubRSA2048Key) || crypto_custom.flag != 3)
{
    Console.WriteLine("Invalid license. Exitting...");
    setLicenseErrorState();
    return;
}
```

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().



```
if (!crypto_custom.check_license(deviceInfo.deviceHash, licText, pubRSA2048Key) || crypto_custom.flag != 3)
{
    Console.WriteLine("Invalid license. Exitting...");
    setLicenseErrorState();
    return;
}
```



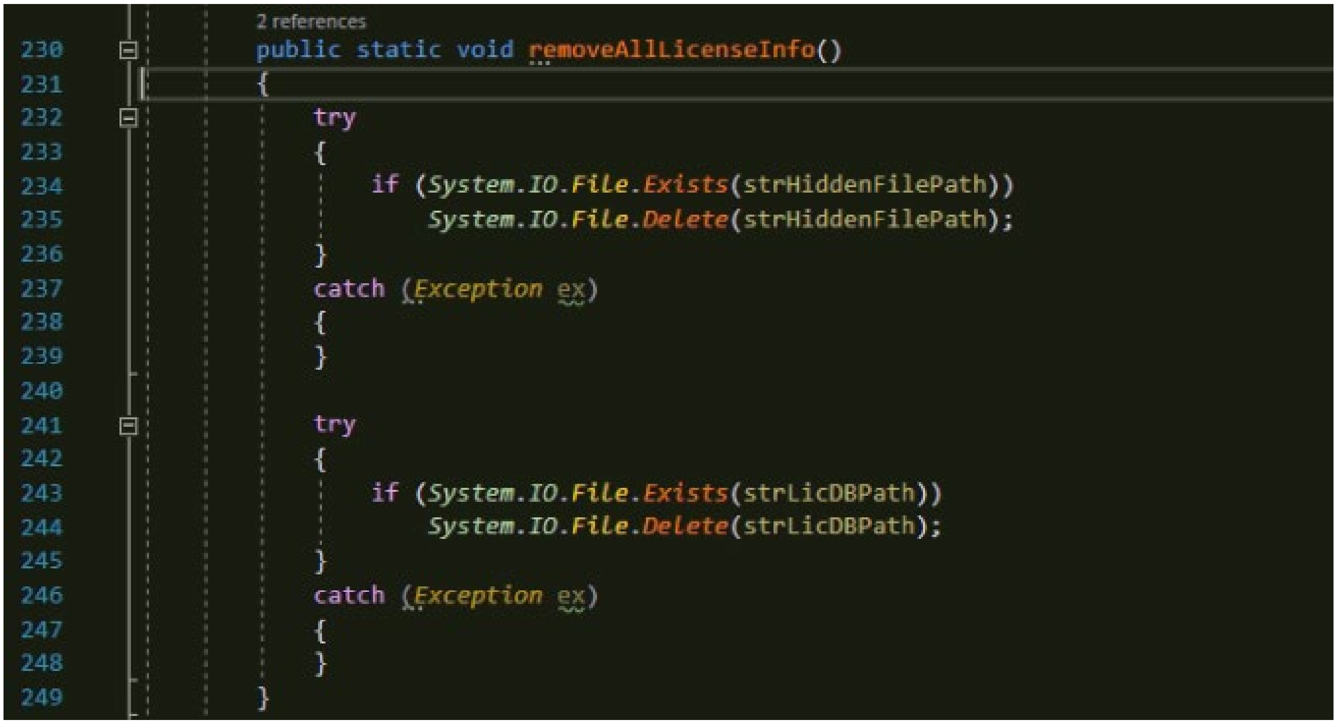
```
7 references
private static void setLicenseErrorState()
{
    if (licTimer != null)
        licTimer.Close();
    removeAllLicenseInfo();
    //Console.ReadKey();
    System.Environment.Exit(0);
}
```

Close timer that is running.

Remove all licensing info.

Close .net application.

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



```
2 references
public static void removeAllLicenseInfo()
{
    try
    {
        if (System.IO.File.Exists(strHiddenFilePath))
            System.IO.File.Delete(strHiddenFilePath);
    }
    catch (Exception ex)
    {
    }

    try
    {
        if (System.IO.File.Exists(strLicDBPath))
            System.IO.File.Delete(strLicDBPath);
    }
    catch (Exception ex)
    {
    }
}
```

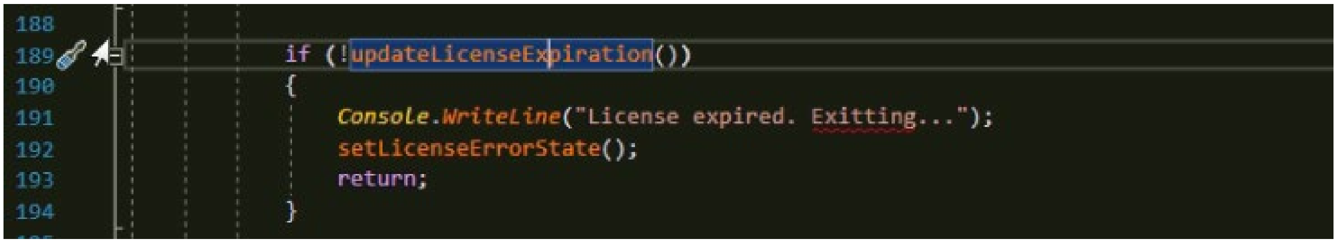
- *Expiration check*

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.

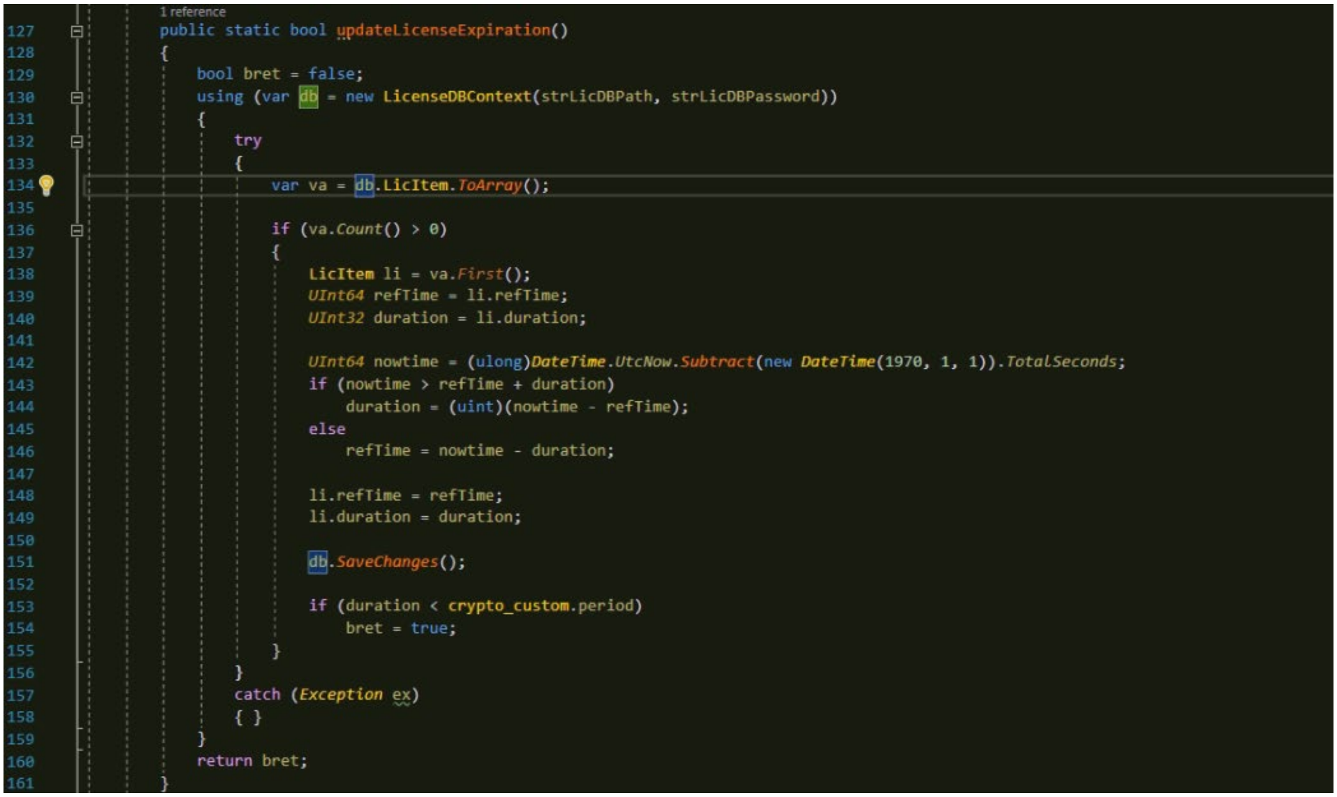
“Issued at” <= now <= “issued at” + “period”

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



```
if (!updateLicenseExpiration())
{
    Console.WriteLine("License expired. Exitting...");
    setLicenseErrorState();
    return;
}
```

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

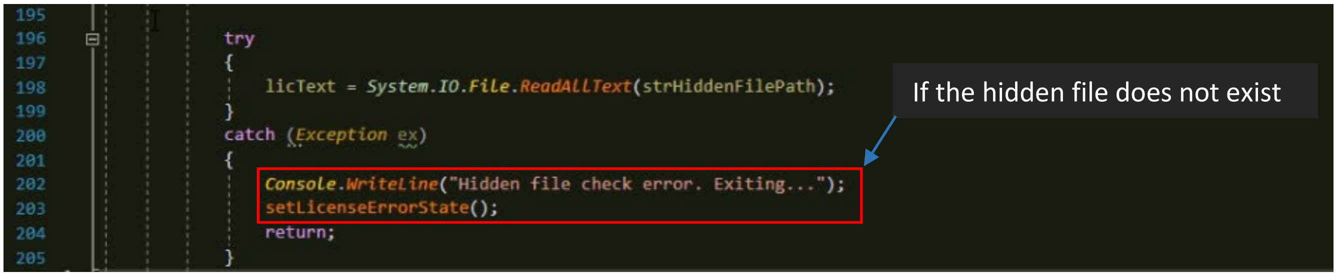


```
1 reference
public static bool updateLicenseExpiration()
{
    bool bret = false;
    using (var db = new LicenseDBContext(strLicDBPath, strLicDBPassword))
    {
        try
        {
            var va = db.LicItem.ToArray();

            if (va.Count() > 0)
            {
                LicItem li = va.First();
                UInt64 refTime = li.refTime;
                UInt32 duration = li.duration;

                UInt64 nowtime = (ulong)DateTime.UtcNow.Subtract(new DateTime(1970, 1, 1)).TotalSeconds;
                if (nowtime > refTime + duration)
                    duration = (uint)(nowtime - refTime);
                else
                    refTime = nowtime - duration;

                li.refTime = refTime;
                li.duration = duration;

                db.SaveChanges();

                if (duration < crypto_custom.period)
                    bret = true;
            }
        }
        catch (Exception ex)
        { }
    }
    return bret;
}
```

- *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.

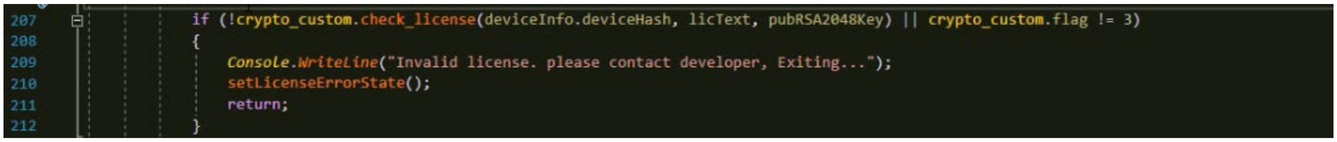


```

try
{
    licText = System.IO.File.ReadAllText(strHiddenFilePath);
}
catch (Exception ex)
{
    Console.WriteLine("Hidden file check error. Exiting...");
    setLicenseErrorState();
    return;
}
```

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.”

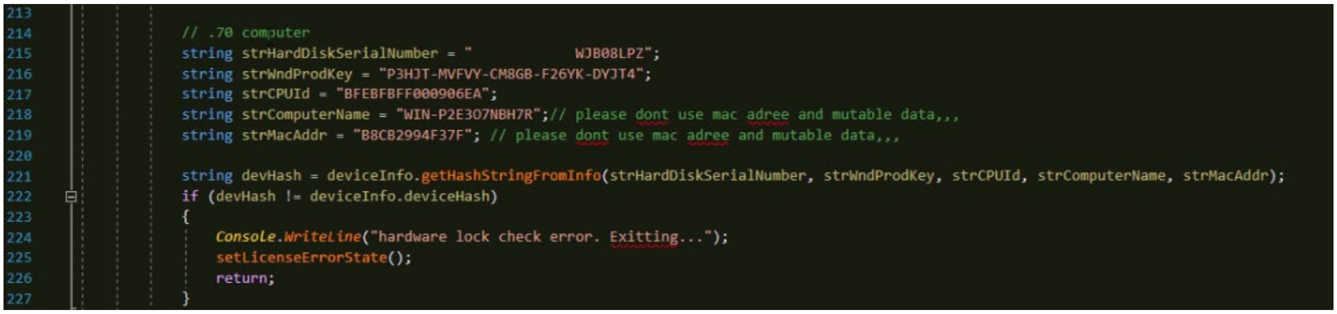


```
if (!crypto_custom.check_license(deviceInfo.deviceHash, licText, pubRSA2048Key) || crypto_custom.flag != 3)
{
    Console.WriteLine("Invalid license. please contact developer, Exiting...");
    setLicenseErrorState();
    return;
}
```

- *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.



```
// .70 computer
string strHardDiskSerialNumber = "            WJB08LPZ";
string strWndProdKey = "P3HJT-MVFVY-CM8GB-F26YK-DYJT4";
string strCPUId = "BFEBFBFF000906EA";
string strComputerName = "WIN-P2E307NBH7R";// please dont use mac adree and mutable data,,,
string strMacAddr = "B8CB2994F37F"; // please dont use mac adree and mutable data,,,

string devHash = deviceInfo.getHashStringFromInfo(strHardDiskSerialNumber, strWndProdKey, strCPUId, strComputerName, strMacAddr);
if (devHash != deviceInfo.deviceHash)
{
    Console.WriteLine("hardware lock check error. Exitting...");
    setLicenseErrorState();
    return;
}
```

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

4. External resources used for .net licensing

1) Folder

No folder used.

2) File

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

3) Registry

The registry key “HKEY_LOCAL_MACHINE/SOFTWARE/Microsoft/Windows NT/CurrentVersion” is used for retrieving windows product id.

4) Hardware

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

5. FAQ

- *How to bar hardware information from composition of device id string*
- *How to change the period of license check*