sha1.cm

```
/*
    ====================================================

    Copyright (c) 2012−2016 Seppo Laakko
    http://sourceforge.net/projects/cmajor/

    Distributed under the GNU General Public License, version 3 (GPLv3).
    (See accompanying LICENSE.txt or http://www.gnu.org/licenses/gpl.html
       )

    ====================================================

    */
// Copyright (c) 1994
// Hewlett−Packard Company
// Copyright (c) 1996
// Silicon Graphics Computer Systems, Inc.
// Copyright (c) 2009 Alexander Stepanov and Paul McJones

using System.IO;

namespace System.Security
{
    public nothrow inline uint LeftRotate(uint x, uint n)
    {
        return (x << n) ^ (x >> (32u − n));
    }

    public class Sha1
    {
        public nothrow Sha1()
        {
            Reset();
        }
        public nothrow void Reset()
        {
            digest[0] = 0x67452301u;
            digest[1] = 0xEFCDAB89u;
            digest[2] = 0x98BADCFEu;
            digest[3] = 0x10325476u;
            digest[4] = 0xC3D2E1F0u;
            byteIndex = 0u;
            bitCount = 0u;
        }
        public nothrow void Process(byte x)
        {
            ProcessByte(x);
            bitCount = bitCount + 8u;
        }
        public nothrow void Process(const void* begin, const void* end)
        {
```

```
        byte* b = cast<byte*>(begin);
        byte* e = cast<byte*>(end);
        while (b != e)
        {
            Process(*b);
            ++b;
        }
    }
    public nothrow void Process(const void* buf, int count)
    {
        byte* b = cast<byte*>(buf);
        Process(b, b + count);
    }
    public nothrow string GetDigest()
    {
        ProcessByte(0x80u);
        if (byteIndex > 56u)
        {
            while (byteIndex != 0u)
            {
                ProcessByte(0u);
            }
            while (byteIndex < 56u)
            {
                ProcessByte(0u);
            }
        }
        else
        {
            while (byteIndex < 56u)
            {
                ProcessByte(0u);
            }
        }
        ProcessByte(cast<byte>((bitCount >> 56u) & 0xFFu));
        ProcessByte(cast<byte>((bitCount >> 48u) & 0xFFu));
        ProcessByte(cast<byte>((bitCount >> 40u) & 0xFFu));
        ProcessByte(cast<byte>((bitCount >> 32u) & 0xFFu));
        ProcessByte(cast<byte>((bitCount >> 24u) & 0xFFu));
        ProcessByte(cast<byte>((bitCount >> 16u) & 0xFFu));
        ProcessByte(cast<byte>((bitCount >> 8u) & 0xFFu));
        ProcessByte(cast<byte>(bitCount & 0xFFu));
        string s = ToHexString(digest[0]);
        s.Append(ToHexString(digest[1]));
        s.Append(ToHexString(digest[2]));
        s.Append(ToHexString(digest[3]));
        s.Append(ToHexString(digest[4]));
        return s;
    }
    private nothrow void ProcessByte(byte x)
    {
        block[byteIndex++] = x;
        if (byteIndex == 64u)
```

```
            {
                byteIndex = 0u;
                ProcessBlock();
            }
        }
    private nothrow void ProcessBlock()
    {
        uint[80] w;
        for (int i = 0; i < 16; ++i)
        {
            w[i] = cast<uint>(block[4 * i]) << 24u;
            w[i] = w[i] | cast<uint>(block[4 * i + 1]) << 16u;
            w[i] = w[i] | cast<uint>(block[4 * i + 2]) << 8u;
            w[i] = w[i] | cast<uint>(block[4 * i + 3]);
        }
        for (int i = 16; i < 80; ++i)
        {
            w[i] = LeftRotate(w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i -
                16], 1u);
        }
        uint a = digest[0];
        uint b = digest[1];
        uint c = digest[2];
        uint d = digest[3];
        uint e = digest[4];
        for (int i = 0; i < 80; ++i)
        {
            uint f;
            uint k;
            if (i < 20)
            {
                f = (b & c) | (~b & d);
                k = 0x5A827999u;
            }
            else if (i < 40)
            {
                f = b ^ c ^ d;
                k = 0x6ED9EBA1u;
            }
            else if (i < 60)
            {
                f = (b & c) | (b & d) | (c & d);
                k = 0x8F1BBCDCu;
            }
            else
            {
                f = b ^ c ^ d;
                k = 0xCA62C1D6u;
            }
            uint temp = LeftRotate(a, 5u) + f + e + k + w[i];
            e = d;
            d = c;
            c = LeftRotate(b, 30u);
```

```
                    b = a;
                    a = temp;
                }
                digest[0] = digest[0] + a;
                digest[1] = digest[1] + b;
                digest[2] = digest[2] + c;
                digest[3] = digest[3] + d;
                digest[4] = digest[4] + e;
            }
            private uint[5] digest;
            private byte[64] block;
            private byte byteIndex;
            private ulong bitCount;
        }

        public nothrow string GetSha1MessageDigest(const string& message)
        {
            Sha1 sha1;
            sha1.Process(message.Chars(), message.Length());
            return sha1.GetDigest();
        }

        public string GetSha1FileDigest(const string& filePath)
        {
            Sha1 sha1;
            BinaryFileStream file(filePath, OpenMode.readOnly);
            IOBuffer buffer(4096u);
            int bytesRead = file.Read(buffer.Mem(), buffer.Size());
            while (bytesRead > 0)
            {
                sha1.Process(buffer.Mem(), bytesRead);
                bytesRead = file.Read(buffer.Mem(), buffer.Size());
            }
            return sha1.GetDigest();
        }
}
```