sha256.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 RightRotate(uint x, uint n)
    {
        return (x >> n) | (x << (32u - n));
    }

    public class Sha256
    {
        static nothrow Sha256()
        {
            k[0] = 0x428a2f98u;
            k[1] = 0x71374491u;
            k[2] = 0xb5c0fbcfu;
            k[3] = 0xe9b5dba5u;
            k[4] = 0x3956c25bu;
            k[5] = 0x59f111f1u;
            k[6] = 0x923f82a4u;
            k[7] = 0xab1c5ed5u;
            k[8] = 0xd807aa98u;
            k[9] = 0x12835b01u;
            k[10] = 0x243185beu;
            k[11] = 0x550c7dc3u;
            k[12] = 0x72be5d74u;
            k[13] = 0x80deb1feu;
            k[14] = 0x9bdc06a7u;
            k[15] = 0xc19bf174u;
            k[16] = 0xe49b69c1u;
            k[17] = 0xefbe4786u;
            k[18] = 0x0fc19dc6u;
```

```
        k[19] = 0x240ca1ccu;
        k[20] = 0x2de92c6fu;
        k[21] = 0x4a7484aau;
        k[22] = 0x5cb0a9dcu;
        k[23] = 0x76f988dau;
        k[24] = 0x983e5152u;
        k[25] = 0xa831c66du;
        k[26] = 0xb00327c8u;
        k[27] = 0xbf597fc7u;
        k[28] = 0xc6e00bf3u;
        k[29] = 0xd5a79147u;
        k[30] = 0x06ca6351u;
        k[31] = 0x14292967u;
        k[32] = 0x27b70a85u;
        k[33] = 0x2e1b2138u;
        k[34] = 0x4d2c6dfcu;
        k[35] = 0x53380d13u;
        k[36] = 0x650a7354u;
        k[37] = 0x766a0abbu;
        k[38] = 0x81c2c92eu;
        k[39] = 0x92722c85u;
        k[40] = 0xa2bfe8a1u;
        k[41] = 0xa81a664bu;
        k[42] = 0xc24b8b70u;
        k[43] = 0xc76c51a3u;
        k[44] = 0xd192e819u;
        k[45] = 0xd6990624u;
        k[46] = 0xf40e3585u;
        k[47] = 0x106aa070u;
        k[48] = 0x19a4c116u;
        k[49] = 0x1e376c08u;
        k[50] = 0x2748774cu;
        k[51] = 0x34b0bcb5u;
        k[52] = 0x391c0cb3u;
        k[53] = 0x4ed8aa4au;
        k[54] = 0x5b9cca4fu;
        k[55] = 0x682e6ff3u;
        k[56] = 0x748f82eeu;
        k[57] = 0x78a5636fu;
        k[58] = 0x84c87814u;
        k[59] = 0x8cc70208u;
        k[60] = 0x90befffau;
        k[61] = 0xa4506cebu;
        k[62] = 0xbef9a3f7u;
        k[63] = 0xc67178f2u;
    }
    public nothrow Sha256()
    {
        Reset();
    }
    public nothrow void Reset()
    {
        digest[0] = 0x6a09e667u;
```

```
            digest[1] = 0xbb67ae85u;
            digest[2] = 0x3c6ef372u;
            digest[3] = 0xa54ff53au;
            digest[4] = 0x510e527fu;
            digest[5] = 0x9b05688cu;
            digest[6] = 0x1f83d9abu;
            digest[7] = 0x5be0cd19u;
        }
        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]));
        s.Append(ToHexString(digest[5]));
        s.Append(ToHexString(digest[6]));
        s.Append(ToHexString(digest[7]));
        return s;
    }
    private nothrow void ProcessByte(byte x)
    {
        block[byteIndex++] = x;
        if (byteIndex == 64u)
        {
            byteIndex = 0u;
            ProcessBlock();
        }
    }
    private nothrow void ProcessBlock()
    {
        uint[64] 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 < 64; ++i)
        {
            w[i] = (RightRotate(w[i - 2], 17u) ^ RightRotate(w[i -
                2], 19u) ^ (w[i - 2] >> 10u)) + w[i - 7] + (
                RightRotate(w[i - 15], 7u) ^ RightRotate(w[i - 15], 18
                u) ^ (w[i - 15] >> 3u)) + w[i - 16];
        }
        uint a = digest[0];
        uint b = digest[1];
        uint c = digest[2];
        uint d = digest[3];
        uint e = digest[4];
        uint f = digest[5];
        uint g = digest[6];
        uint h = digest[7];
        for (int i = 0; i < 64; ++i)
        {
            uint t1 = h + (RightRotate(e, 6u) ^ RightRotate(e, 11u) ^
                RightRotate(e, 25u)) + ((e & f) ^ ((~e) & g)) + k[i]
               + w[i];
```

```
                uint t2 = (RightRotate(a, 2u) ^ RightRotate(a, 13u) ^
                    RightRotate(a, 22u)) + ((a & b) ^ (a & c) ^ (b & c));
                h = g;
                g = f;
                f = e;
                e = d + t1;
                d = c;
                c = b;
                b = a;
                a = t1 + t2;
            }
            digest[0] = a + digest[0];
            digest[1] = b + digest[1];
            digest[2] = c + digest[2];
            digest[3] = d + digest[3];
            digest[4] = e + digest[4];
            digest[5] = f + digest[5];
            digest[6] = g + digest[6];
            digest[7] = h + digest[7];
        }
        private static uint[64] k;
        private byte[64] block;
        private byte byteIndex;
        private ulong bitCount;
        private uint[8] digest;
    }

    public nothrow string GetSha256MessageDigest(const string& message)
    {
        Sha256 sha256;
        sha256.Process(message.Chars(), message.Length());
        return sha256.GetDigest();
    }

    public string GetSha256FileDigest(const string& filePath)
    {
        Sha256 sha256;
        BinaryFileStream file(filePath, OpenMode.readOnly);
        IOBuffer buffer(4096u);
        int bytesRead = file.Read(buffer.Mem(), buffer.Size());
        while (bytesRead > 0)
        {
            sha256.Process(buffer.Mem(), bytesRead);
            bytesRead = file.Read(buffer.Mem(), buffer.Size());
        }
        return sha256.GetDigest();
    }
}
```