# MD5 Hash

```
//http://facebooktoolkit.codeplex.com/
//http://facebooktoolkit.codeplex.com/license
//Copyright (c) Microsoft Corporation. All rights reserved.
using System;
using System.Text;

namespace Facebook.Utility
{
internal static class MD5Core
{
//Prevent CSC from adding a default public constructor

public static byte[] GetHash(string input, Encoding encoding)
{
if (null == input)
throw new ArgumentNullException(“input”, “Unable to calculate hash over null input data”);
if (null == encoding)
throw new ArgumentNullException(“encoding”, “Unable to calculate hash over a string without a default encoding. Consider using the GetHash(string) overload to use UTF8 Encoding”);

byte[] target = encoding.GetBytes(input);

return GetHash(target);
}

public static byte[] GetHash(string input)
{
return GetHash(input, new UTF8Encoding());
}

public static string GetHashString(byte[] input)
{
```

```
if (null == input)
throw new ArgumentNullException(“input”, “Unable to calculate hash over null input data”);

string retval = BitConverter.ToString(GetHash(input));
retval = retval.Replace(“-“, “”);

return retval;
}

public static string GetHashString(string input, Encoding encoding)
{
if (null == input)
throw new ArgumentNullException(“input”, “Unable to calculate hash over null input data”);
if (null == encoding)
throw new ArgumentNullException(“encoding”, “Unable to calculate hash over a string without a default encoding. Consider using the GetHashString(string) overload to use UTF8 Encoding”);

byte[] target = encoding.GetBytes(input);

return GetHashString(target);
}

public static string GetHashString(string input)
{
return GetHashString(input, new UTF8Encoding());
}

public static byte[] GetHash(byte[] input)
{
//Intitial values defined in RFC 1321
uint a = 0x67452301;
uint b = 0xefcdab89;
uint c = 0x98badcfe;
uint d = 0x10325476;
```

```
if (null == input)
throw new ArgumentNullException(“input”, “Unable to calculate hash over null input data”);

//Calculate the number of 64-byte blocks required to be hashed, rounded down. Include 8 bytes for padding & length appending.
int chunks = (input.Length + 8) / 64;

//Remainder is the offset (0 indexed) in the last chunk where padding begins.
//NOTE: In the event that padding starts in the second to last chunk, remainder can be negative
int remainder = input.Length – (chunks * 64);

//CHUNK IMPLEMENTATION
for (int i = 0; i <= chunks; i++) { uint[] temp; byte[] working = new byte[64]; //Last chunk if (i == chunks) { //This chunk contains both data and padding if (0 <= remainder) { //Copy all reamaining data. Array.Copy(input, i * 64, working, 0, remainder); //Padding is a single bit 1, followed by the number of 0s required to make size congruent to 448 modulo 512. Step 1 of RFC 1321 working[remainder] = 0x80; } //The CLR ensures that our buffer is 0-assigned, we don't need to explicitly set it //Input length in bits can be larger then a 32 bit integer. UInt64 l = (UInt64)input.Length; l *= 8; byte[] length = BitConverter.GetBytes(l); //Length in BITS is added to the end of the array. Step 2 of RFC 1321 Array.Copy(length, 0, working, 56, 8); temp = Converter(working); } //Second to last chunk, but our chunks are such that this one contains some padding else if ((0 > remainder) && ((i + 1) == chunks))
{
Array.Copy(input, i * 64, working, 0, 64 + remainder);
working[64 + remainder] = 0x80;
//The CLR ensures that our buffer is 0-assigned, we don't need to explicitly set it
temp = Converter(working);
```

```
}
//Only the last chunk contains padding, or we're not yet at
the second to last chunk
else
{
Array.Copy(input, i * 64, working, 0, 64);
temp = Converter(working);
}

//Defining these inside the scope actualy nets us a small perf
improvement.
uint aa = a;
uint bb = b;
uint cc = c;
uint dd = d;

a = r1(a, b, c, d, temp[0], 7, 0xd76aa478);
d = r1(d, a, b, c, temp[1], 12, 0xe8c7b756);
c = r1(c, d, a, b, temp[2], 17, 0x242070db);
b = r1(b, c, d, a, temp[3], 22, 0xc1bdceee);
a = r1(a, b, c, d, temp[4], 7, 0xf57c0faf);
d = r1(d, a, b, c, temp[5], 12, 0x4787c62a);
c = r1(c, d, a, b, temp[6], 17, 0xa8304613);
b = r1(b, c, d, a, temp[7], 22, 0xfd469501);
a = r1(a, b, c, d, temp[8], 7, 0x698098d8);
d = r1(d, a, b, c, temp[9], 12, 0x8b44f7af);
c = r1(c, d, a, b, temp[10], 17, 0xffff5bb1);
b = r1(b, c, d, a, temp[11], 22, 0x895cd7be);
a = r1(a, b, c, d, temp[12], 7, 0x6b901122);
d = r1(d, a, b, c, temp[13], 12, 0xfd987193);
c = r1(c, d, a, b, temp[14], 17, 0xa679438e);
b = r1(b, c, d, a, temp[15], 22, 0x49b40821);

a = r2(a, b, c, d, temp[1], 5, 0xf61e2562);
d = r2(d, a, b, c, temp[6], 9, 0xc040b340);
c = r2(c, d, a, b, temp[11], 14, 0x265e5a51);
b = r2(b, c, d, a, temp[0], 20, 0xe9b6c7aa);
a = r2(a, b, c, d, temp[5], 5, 0xd62f105d);
```

```
d = r2(d, a, b, c, temp[10], 9, 0x02441453);
c = r2(c, d, a, b, temp[15], 14, 0xd8a1e681);
b = r2(b, c, d, a, temp[4], 20, 0xe7d3fbc8);
a = r2(a, b, c, d, temp[9], 5, 0x21e1cde6);
d = r2(d, a, b, c, temp[14], 9, 0xc33707d6);
c = r2(c, d, a, b, temp[3], 14, 0xf4d50d87);
b = r2(b, c, d, a, temp[8], 20, 0x455a14ed);
a = r2(a, b, c, d, temp[13], 5, 0xa9e3e905);
d = r2(d, a, b, c, temp[2], 9, 0xfcefa3f8);
c = r2(c, d, a, b, temp[7], 14, 0x676f02d9);
b = r2(b, c, d, a, temp[12], 20, 0x8d2a4c8a);

a = r3(a, b, c, d, temp[5], 4, 0xfffa3942);
d = r3(d, a, b, c, temp[8], 11, 0x8771f681);
c = r3(c, d, a, b, temp[11], 16, 0x6d9d6122);
b = r3(b, c, d, a, temp[14], 23, 0xfde5380c);
a = r3(a, b, c, d, temp[1], 4, 0xa4beea44);
d = r3(d, a, b, c, temp[4], 11, 0x4bdecfa9);
c = r3(c, d, a, b, temp[7], 16, 0xf6bb4b60);
b = r3(b, c, d, a, temp[10], 23, 0xbebfbc70);
a = r3(a, b, c, d, temp[13], 4, 0x289b7ec6);
d = r3(d, a, b, c, temp[0], 11, 0xeaa127fa);
c = r3(c, d, a, b, temp[3], 16, 0xd4ef3085);
b = r3(b, c, d, a, temp[6], 23, 0x04881d05);
a = r3(a, b, c, d, temp[9], 4, 0xd9d4d039);
d = r3(d, a, b, c, temp[12], 11, 0xe6db99e5);
c = r3(c, d, a, b, temp[15], 16, 0x1fa27cf8);
b = r3(b, c, d, a, temp[2], 23, 0xc4ac5665);

a = r4(a, b, c, d, temp[0], 6, 0xf4292244);
d = r4(d, a, b, c, temp[7], 10, 0x432aff97);
c = r4(c, d, a, b, temp[14], 15, 0xab9423a7);
b = r4(b, c, d, a, temp[5], 21, 0xfc93a039);
a = r4(a, b, c, d, temp[12], 6, 0x655b59c3);
d = r4(d, a, b, c, temp[3], 10, 0x8f0ccc92);
c = r4(c, d, a, b, temp[10], 15, 0xffeff47d);
b = r4(b, c, d, a, temp[1], 21, 0x85845dd1);
```

```
a = r4(a, b, c, d, temp[8], 6, 0x6fa87e4f);
d = r4(d, a, b, c, temp[15], 10, 0xfe2ce6e0);
c = r4(c, d, a, b, temp[6], 15, 0xa3014314);
b = r4(b, c, d, a, temp[13], 21, 0x4e0811a1);
a = r4(a, b, c, d, temp[4], 6, 0xf7537e82);
d = r4(d, a, b, c, temp[11], 10, 0xbd3af235);
c = r4(c, d, a, b, temp[2], 15, 0x2ad7d2bb);
b = r4(b, c, d, a, temp[9], 21, 0xeb86d391);

a = unchecked(a + aa);
b = unchecked(b + bb);
c = unchecked(c + cc);
d = unchecked(d + dd);
}

//Output
byte[] output = new byte[16];
Array.Copy(BitConverter.GetBytes(a), 0, output, 0, 4);
Array.Copy(BitConverter.GetBytes(b), 0, output, 4, 4);
Array.Copy(BitConverter.GetBytes(c), 0, output, 8, 4);
Array.Copy(BitConverter.GetBytes(d), 0, output, 12, 4);

return output;
}

//Manually unrolling these equations nets us a 20% performance
improvement
private static uint r1(uint a, uint b, uint c, uint d, uint x,
int s, uint t)
{
// (b + LSR((a + F(b, c, d) + x + t), s))
//F(x, y, z) ((x & y) | ((x ^ 0xFFFFFFFF) & z))
return unchecked(b + LSR((a + ((b & c) | ((b ^ 0xFFFFFFFF) &
d)) + x + t), s));
}

private static uint r2(uint a, uint b, uint c, uint d, uint x,
int s, uint t)
```

```
{
// (b + LSR((a + G(b, c, d) + x + t), s))
//G(x, y, z) ((x & z) | (y & (z ^ 0xFFFFFFFF)))
return unchecked(b + LSR((a + ((b & d) | (c & (d ^
0xFFFFFFFF))) + x + t), s));
}

private static uint r3(uint a, uint b, uint c, uint d, uint x,
int s, uint t)
{
// (b + LSR((a + H(b, c, d) + k + i), s))
//H(x, y, z) (x ^ y ^ z)
return unchecked(b + LSR((a + (b ^ c ^ d) + x + t), s));
}

private static uint r4(uint a, uint b, uint c, uint d, uint x,
int s, uint t)
{
// (b + LSR((a + I(b, c, d) + k + i), s))
//I(x, y, z) (y ^ (x | (z ^ 0xFFFFFFFF)))
return unchecked(b + LSR((a + (c ^ (b | (d ^ 0xFFFFFFFF))) + x
+ t), s));
}

// Implementation of left rotate
// s is an int instead of a uint becuase the CLR requires the
argument passed to >>/<> (32 – s)));
}

//Convert input array into array of UInts
private static uint[] Converter(byte[] input)
{
if (null == input)
throw new ArgumentNullException(“input”, “Unable convert null
array to array of uInts”);

if (input.Length != 64)
throw new ArgumentException(“Input length must be 64 bytes”,
```

```
“input”);

uint[] result = new uint[16];

for (int i = 0; i < 16; i++) { result[i] = input[i * 4];
result[i] += (uint)input[i * 4 + 1] <
```