Base64 Encoder

```
//http://www.bouncycastle.org/
//MIT X11 License
using System;
using System.IO;

namespace Org.BouncyCastle.Utilities.Encoders
{
public class Base64Encoder

{
protected readonly byte[] encodingTable =
{
(byte)'A',  (byte)'B',  (byte)'C',  (byte)'D',  (byte)'E',
(byte)'F', (byte)'G',
(byte)'H',  (byte)'I',  (byte)'J',  (byte)'K',  (byte)'L',
(byte)'M', (byte)'N',
(byte)'O',  (byte)'P',  (byte)'Q',  (byte)'R',  (byte)'S',
(byte)'T', (byte)'U',
(byte)'V', (byte)'W', (byte)'X', (byte)'Y', (byte)'Z',
(byte)'a',  (byte)'b',  (byte)'c',  (byte)'d',  (byte)'e',
(byte)'f', (byte)'g',
(byte)'h',  (byte)'i',  (byte)'j',  (byte)'k',  (byte)'l',
(byte)'m', (byte)'n',
(byte)'o',  (byte)'p',  (byte)'q',  (byte)'r',  (byte)'s',
(byte)'t', (byte)'u',
(byte)'v',
(byte)'w', (byte)'x', (byte)'y', (byte)'z',
(byte)'0',  (byte)'1',  (byte)'2',  (byte)'3',  (byte)'4',
(byte)'5', (byte)'6',
(byte)'7', (byte)'8', (byte)'9',
(byte)'+', (byte)'/'
};

protected byte padding = (byte)'=';
```

```
/*
* set up the decoding table.
*/
protected readonly byte[] decodingTable = new byte[128];

protected void InitialiseDecodingTable()
{
for (int i = 0; i < encodingTable.Length; i++) { decodingTable[encodingTable[i]] = (byte)i; } } public Base64Encoder() { InitialiseDecodingTable(); } /** * encode the input data producing a base 64 output stream. * * @return the number of bytes produced. */ public int Encode( byte[] data, int off, int length, Stream outStream) { int modulus = length % 3; int dataLength = (length - modulus); int a1, a2, a3; for (int i = off; i < off + dataLength; i += 3) { a1 = data[i] & 0xff; a2 = data[i + 1] & 0xff; a3 = data[i + 2] & 0xff; outStream.WriteByte(encodingTable[(int) ((uint) a1 >> 2) & 0x3f]);
outStream.WriteByte(encodingTable[((a1 <> 4)) & 0x3f]);
outStream.WriteByte(encodingTable[((a2 <> 6)) & 0x3f]);
outStream.WriteByte(encodingTable[a3 & 0x3f]);
}

/*
* process the tail end.
*/
int b1, b2, b3;
int d1, d2;

switch (modulus)
{
case 0: /* nothing left to do */
break;
case 1:
d1 = data[off + dataLength] & 0xff;
b1 = (d1 >> 2) & 0x3f;
b2 = (d1 <> 2) & 0x3f;
b2 = ((d1 <> 4)) & 0x3f;
```

```
b3 = (d2 < off)
{
if (!ignore((char)data[end – 1]))
{
break;
}

end–;
}

int i = off;
int finish = end – 4;

i = nextI(data, i, finish);

while (i < finish) { b1 = decodingTable[data[i++]]; i =
nextI(data, i, finish); b2 = decodingTable[data[i++]]; i =
nextI(data, i, finish); b3 = decodingTable[data[i++]]; i =
nextI(data, i, finish); b4 = decodingTable[data[i++]];
outStream.WriteByte((byte)((b1 <> 4)));
outStream.WriteByte((byte)((b2 <> 2)));
outStream.WriteByte((byte)((b3 < 0)
{
if (!ignore(data[end – 1]))
{
break;
}

end–;
}

int i = 0;
int finish = end – 4;

i = nextI(data, i, finish);

while (i < finish) { b1 = decodingTable[data[i++]]; i =
nextI(data, i, finish); b2 = decodingTable[data[i++]]; i =
nextI(data, i, finish); b3 = decodingTable[data[i++]]; i =
```

```
nextI(data, i, finish); b4 = decodingTable[data[i++]];
outStream.WriteByte((byte)((b1 <> 4)));
outStream.WriteByte((byte)((b2 <> 2)));
outStream.WriteByte((byte)((b3 <> 4)));

return 1;
}

if (c4 == padding)
{
byte b1 = decodingTable[c1];
byte b2 = decodingTable[c2];
byte b3 = decodingTable[c3];

outStream.WriteByte((byte)((b1 <> 4)));
outStream.WriteByte((byte)((b2 <> 2)));

return 2;
}

{
byte b1 = decodingTable[c1];
byte b2 = decodingTable[c2];
byte b3 = decodingTable[c3];
byte b4 = decodingTable[c4];

outStream.WriteByte((byte)((b1 <> 4)));
outStream.WriteByte((byte)((b2 <> 2)));
outStream.WriteByte((byte)((b3 <
```