Decodes all or part of the input base64 encoded StringBuffer

```
/*****************************************************************
*****************
* The MIT License
* Copyright (c) 2003 Novell Inc. www.novell.com
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the Software), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED AS IS, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
```

```
OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR
OTHER DEALINGS IN THE
* SOFTWARE.
**********************************************************
*****************/
//
// Novell.Directory.Ldap.Utilclass.Base64.cs
//
// Author:
// Sunil Kumar (Sunilk@novell.com)
//
// (C) 2003 Novell, Inc (http://www.novell.com)
//

using System;

namespace Novell.Directory.Ldap.Utilclass
{

///

The Base64 utility class performs base64 encoding and
decoding.
///
/// The Base64 Content-Transfer-Encoding is designed to
represent
/// arbitrary sequences of octets in a form that need not be
humanly
/// readable. The encoding and decoding algorithms are simple,
but the
/// encoded data are consistently only about 33 percent larger
than the
/// unencoded data. The base64 encoding algorithm is defined
by
/// RFC 2045.
///
public class Base64
{
///
```

```
Decodes a base64 encoded StringBuffer.
/// Decodes all or part of the input base64 encoded
StringBuffer, each
/// Character value representing a base64 character. The
resulting
/// binary data is returned as an array of bytes.
///
///
/// The StringBuffer object that contains base64
/// encoded data.
///  ///  The start index of the base64 encoded data.
///  ///  The end index + 1 of the base64 encoded data.
///
///  ///  The decoded byte array
///
public  static  sbyte[]  decode(System.Text.StringBuilder
encodedSBuf, int start, int end)
{
///

conversion table for decoding from base64.
///
/// dmap is a base64 (8-bit) to six-bit value converstion
table.
/// For example the ASCII character 'P' has a value of 80.
/// The value in the 80th position of the table is 0x0f or 15.
/// 15 is the original 6-bit value that the letter 'P'
represents.
///
/*
* 6-bit decoded value base64 base64
* encoded character
* value
*
* Note: about half of the values in the table are only place
holders
*/
sbyte[] dmap = new sbyte[]{(sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
```

```
(0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x3e),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x3f), (sbyte) (0x34), (sbyte) (0x35), (sbyte) (0x36),
(sbyte) (0x37), (sbyte) (0x38), (sbyte) (0x39), (sbyte)
(0x3a), (sbyte) (0x3b), (sbyte) (0x3c), (sbyte) (0x3d),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x01), (sbyte) (0x02), (sbyte)
(0x03), (sbyte) (0x04), (sbyte) (0x05), (sbyte) (0x06),
(sbyte) (0x07), (sbyte) (0x08), (sbyte) (0x09), (sbyte)
(0x0a), (sbyte) (0x0b), (sbyte) (0x0c), (sbyte) (0x0d),
(sbyte) (0x0e), (sbyte) (0x0f), (sbyte) (0x10), (sbyte)
(0x11), (sbyte) (0x12), (sbyte) (0x13), (sbyte) (0x14),
(sbyte) (0x15), (sbyte) (0x16), (sbyte) (0x17), (sbyte)
(0x18), (sbyte) (0x19), (sbyte) (0x00), (sbyte) (0x00),
(sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00), (sbyte) (0x1a), (sbyte) (0x1b), (sbyte) (0x1c),
(sbyte) (0x1d), (sbyte) (0x1e), (sbyte) (0x1f), (sbyte)
(0x20), (sbyte) (0x21), (sbyte) (0x22), (sbyte) (0x23),
(sbyte) (0x24), (sbyte) (0x25), (sbyte) (0x26), (sbyte)
(0x27), (sbyte) (0x28), (sbyte) (0x29), (sbyte) (0x2a),
(sbyte) (0x2b), (sbyte) (0x2c), (sbyte) (0x2d), (sbyte)
(0x2e), (sbyte) (0x2f), (sbyte) (0x30), (sbyte) (0x31),
(sbyte) (0x32), (sbyte) (0x33), (sbyte) (0x00), (
sbyte) (0x00), (sbyte) (0x00), (sbyte) (0x00), (sbyte)
(0x00)}; // 120-127 'xyz '
```

```
int i, j, k;
int esbLen = end – start; // length of the encoded part
int gn = esbLen / 4; // number of four-bytes group in ebs
int dByteLen; // length of dbs, default is '0'
bool onePad = false, twoPads = false;
sbyte[] decodedBytes; // decoded bytes

if (encodedSBuf.Length == 0)
{
return new sbyte[0];
}
// the number of encoded bytes should be multiple of number 4
if ((esbLen % 4) != 0)
{
throw                                                    new
System.SystemException("Novell.Directory.Ldap.ldif_dsml."  +
"Base64Decoder: decode error: mal-formatted encode value");
}

// every four-bytes in ebs, except the last one if it in the
form of
// three bytes.
if ((encodedSBuf[end – 1] == (int)'=') && (encodedSBuf[end –
2] == (int)'='))
{
// the last four bytes of ebs is in the form of '**=='
twoPads = true;
// the first two bytes of the last four-bytes of ebs will be
// decoded into one byte.
dByteLen = gn * 3 – 2;
decodedBytes = new sbyte[dByteLen];
}
else if (encodedSBuf[end – 1] == (int)'=')
{
// the last four bytes of ebs is in the form of '***='
onePad = true;
// the first two bytes of the last four-bytes of ebs will be
```

```
// decoded into two bytes.
dByteLen = gn * 3 – 1;
decodedBytes = new sbyte[dByteLen];
}
else
{
// the last four bytes of ebs is in the form of '****', eg. no
pad.
dByteLen = gn * 3;
decodedBytes = new sbyte[dByteLen];
}

// map of encoded and decoded bits
// no padding:
// bits in 4 encoded bytes: 76543210 76543210 76543210
76543210
// bits in 3 decoded bytes: 765432 107654 321076 543210
// base64 string “QUFB”:00010000 00010100 000001010 0000001
// plain string “AAA”: 010000 010100 000101 000001
// one padding:
// bits in 4 encoded bytes: 76543210 76543210 76543210
76543210
// bits in 2 decoded bytes: 765432 107654 3210
// base64 string “QUE=”: 00010000 000101000 0000100 00111101
// plain string “AA”: 010000 010100 0001
// two paddings:
// bits in 4 encoded bytes: 76543210 76543210 76543210
76543210
// bits in 1 decoded bytes: 765432 10
// base64 string “QQ==”: 00010000 00010000 00111101 00111101
// plain string “A”: 010000 01
for (i = 0, j = 0, k = 1; i < esbLen; i += 4, j += 3, k++) {
//   build   decodedBytes[j].   decodedBytes[j]   =
(sbyte)(dmap[encodedSBuf[start + i]] <> 4);

// build decodedBytes[j+1]
if ((k == gn) && twoPads)
```

```
{
break;
}
else
{
decodedBytes[j + 1] = (sbyte)((dmap[encodedSBuf[start + i +
1]] & 0x0f) <> 2);
}

// build decodedBytes[j+2]
if ((k == gn) && onePad)
{
break;
}
else
{
decodedBytes[j + 2] = (sbyte)((dmap[encodedSBuf[start + i +
2]] & 0x03) <
```