
Base64 Encoding Torn Apart By Ankit Fadia ankit@bol.net.in

Since its introduction, Base64 encoding has extremely quickly gained popularity. Besides being the default Encoding standard being used for encoding files to be sent as attachments by Multipurpose Internet Mail Extensions or MIME, it has also started being used in a number of other places.

Please note that almost all email clients use MIME to send files as attachments, this in turn means that a majority of email clients are using Base64 to encode files, before being sent across networks.

Another popular usage of Base64 encoding is in the case of Web Servers implementing HTTP Based Basic Authentication. When the server wants to restrict or control the access to certain folders, then, it can password protect them by using HTTP Based Basic Authentication. Basic Authentication uses the Base64 Encoding standard to encode the Username and Password and store them.

So, basically what my point is that Base64 Encoding has a number of practical usages and due to the fact that it is very easy to implement, it is being put to use for a number of purposes at a number of places by a number of parties. Get my point?

However, it continues to remain by far the lamest encoding standard or the poorest means of security. You see instead of the text being passed through a powerful, difficult to break algorithm and being encrypted, it is only being encoded by a relatively simple to reverse encoding standard.

Base64 uses a 65-character subset of US-ASCII, allowing 6 bits for each character. For Example, take the character ‘m’ for instance. The character ‘m’ has a Base64 value of 38.

How did we get this value? Well, there is a Base64 Alphabet chart included at the end of this tutorial, which contains all the alphabets and their corresponding Base64 value. So, each time you want to get the Base64 value of an ASCII character, you need to refer to this Base64 Value chart. Anyway, getting back to our example, the character ‘m’ has a Base64 value of 38, which when represented in binary form, is 100110.

Now, let us take yet another example to see how a text is encoded by Base64 Encoding. Say, that the text to be encoded is: ‘mne’. The text is firstly converted into its decimal value.

The character "m" has the decimal value of 109

The character "n" has the decimal value of 110

The character "e" has the decimal value of 101

This implies that "mne" (three 8-bit-byte text string) is 109 110 101 in decimal form. When converted to binary the string looks like this:

01101101 01101110 01100101

These three 8-bit-bytes are concatenated (linked together) to make a 24-bit stream:

011011010110111001100101

This 24-bit stream is then split up into four 6-bit sections:

011011 010110 111001 100101

We now have 4 values. These binary values, when converted into decimal form look like this:

27 22 57 37

Now each character of the Base64 character set has a decimal value. We now change these decimal values into the Base64 equivalent:

27 = b

22 = w

57 = 5

37 = l

So "mne" when encoded as Base64 reads as "bw5l". Below is a table of the Base64 character set with their decimal values:

Table 1: The Base64 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3

5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

When decoding a Base64 string just do the reverse:

1) Convert the character to its Base64 decimal value.

2) Convert this decimal value into binary.

3) Squash the 6 bits of each character into one big string of binary digits.

4) Split this string up into groups of 8 bits (starting from right to left).

5) Convert each 8-bit binary value into a decimal number.

6) Convert this decimal value into its US-ASCII equivalent.

For those of you who do not want to use the manual method of decoding a Base64 encoded value, I have the following Perl script, which will do it for you:

```
use MIME::Base64;

print decode_base64("Insert Text to be decoded here.");
```

Here's the C source code for the Base 64 encoder/decoder.

```
/*

Dave Winer, dwiner@well.com, UserLand Software, 4/7/97

 */


#include <appletdefs.h>

#include <iac.h>

#include "base64.h"


static char encodingTable [64] = {


   'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P',

   'Q','R','S','T','U','V','W','X','Y','Z','a','b','c','d','e','f',

   'g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v',

   'w','x','y','z','0','1','2','3','4','5','6','7','8','9','+','/'

            };


static unsigned long gethandlesize (Handle h) {
```

```
	return (GetHandleSize (h));
	} /*gethandlesize*/

static boolean sethandlesize (Handle h, unsigned long newsize) {
	 SetHandleSize (h, newsize);
	return (MemError () == noErr);
	} /*sethandlesize*/

static unsigned char gethandlechar (Handle h, unsigned long ix) {

	return ((*h) [ix]);
	} /*gethandlechar*/

static void sethandlechar (Handle h, unsigned long ix, unsigned char ch) {

	(*h) [ix] = ch;
	} /*sethandlechar*/

static boolean encodeHandle (Handle htext, Handle h64, short linelength) {

	/*
	encode the handle. some funny stuff about linelength -- it only makes
```

```
sense to make it a multiple of 4. if it's not a multiple of 4, we make it
so (by only checking it every 4 characters.
further, if it's 0, we don't add any line breaks at all.
*/

unsigned long ixtext;
unsigned long lentext;
unsigned long origsize;
long ctremaining;
unsigned char ch;
unsigned char inbuf [3], outbuf [4];
short i;
short charsonline = 0, ctcopy;
ixtext = 0;
lentext = gethandlesize (htext);
while (true) {

        ctremaining = lentext - ixtext;
                if (ctremaining <= 0)
                break;

        for (i = 0; i < 3; i++) {
                        unsigned long ix = ixtext + i;
```

```
                    if (ix < lentext)
                    inbuf [i] = gethandlechar (htext, ix);
            else
                    inbuf [i] = 0;
            } /*for*/

outbuf [0] = (inbuf [0] & 0xFC) >> 2;
outbuf [1] = ((inbuf [0] & 0x03) << 4) | ((inbuf [1] & 0xF0) >> 4);
outbuf [2] = ((inbuf [1] & 0x0F) << 2) | ((inbuf [2] & 0xC0) >> 6);
outbuf [3] = inbuf [2] & 0x3F;
origsize = gethandlesize (h64);

if (!sethandlesize (h64, origsize + 4))
            return (false);

ctcopy = 4;
switch (ctremaining) {
            case 1:
                    ctcopy = 2;
                    break;

            case 2:
                    ctcopy = 3;
```

```
                        break;
                    } /*switch*/

                for (i = 0; i < ctcopy; i++)
                    sethandlechar (h64, origsize + i, encodingTable [outbuf
[i]]);
                for (i = ctcopy; i < 4; i++)
                    sethandlechar (h64, origsize + i, '=');

                ixtext += 3;
                charsonline += 4;
                if (linelength > 0) { /*DW 4/8/97 -- 0 means no line breaks*/
                        if (charsonline >= linelength) {

                        charsonline = 0;
                        origsize = gethandlesize (h64);

                        if (!sethandlesize (h64, origsize + 1))
                                return (false);
                        sethandlechar (h64, origsize, '\n');
                        }
                    }
                } /*while*/
```

```
	return (true);
	} /*encodeHandle*/

static boolean decodeHandle (Handle h64, Handle htext) {

	unsigned long ixtext;
	unsigned long lentext;
	unsigned long origsize;
	unsigned long ctremaining;
	unsigned char ch;
	unsigned char inbuf [3], outbuf [4];
	short i, ixinbuf;
	boolean flignore;
	boolean flendtext = false;
	ixtext = 0;
	lentext = gethandlesize (h64);
	ixinbuf = 0;
	while (true) {

		if (ixtext >= lentext)
			break;

		ch = gethandlechar (h64, ixtext++);
```

```
flignore = false;


if ((ch >= 'A') && (ch <= 'Z'))
        ch = ch - 'A';
else if ((ch >= 'a') && (ch <= 'z'))
        ch = ch - 'a' + 26;
else if ((ch >= '0') && (ch <= '9'))
        ch = ch - '0' + 52;
else if (ch == '+')
        ch = 62;
else if (ch == '=') /*no op -- can't ignore this one*/
        flendtext = true;
else if (ch == '/')
        ch = 63;
else
        flignore = true;


if (!flignore) {


        short ctcharsinbuf = 3;
        boolean flbreak = false;
         if (flendtext) {
```

```
                    if (ixinbuf == 0)
                            break;
                    if ((ixinbuf == 1) || (ixinbuf == 2))
                            ctcharsinbuf = 1;
                    else
                            ctcharsinbuf = 2;

                    ixinbuf = 3;

                    flbreak = true;
                    }
            inbuf [ixinbuf++] = ch;
            if (ixinbuf == 4) {
                    ixinbuf = 0;
                    outbuf [0] = (inbuf [0] << 2) | ((inbuf [1] & 0x30)
>> 4);
                    outbuf [1] = ((inbuf [1] & 0x0F) << 4) | ((inbuf
[2] & 0x3C) >> 2);
                    outbuf [2] = ((inbuf [2] & 0x03) << 6) | (inbuf [3]
& 0x3F);
                    origsize = gethandlesize (htext);

                    if (!sethandlesize (htext, origsize + ctcharsinbuf))
                            return (false);
                    for (i = 0; i < ctcharsinbuf; i++)
```

```
                                                    sethandlechar (htext, origsize + i,
outbuf [i]);
                                                }
                                        if (flbreak)
                                                break;
                                        }
                                } /*while*/


                exit:
                return (true);
                } /*decodeHandle*/



void base64encodeVerb (void) {
                Handle h64, htext;
                short linelength;


                if (!IACgettextparam ((OSType) keyDirectObject, &htext))
                        return;
                if (!IACgetshortparam ((OSType) 'line', &linelength))
                        return;
                h64 = NewHandle (0);
                if (!encodeHandle (htext, h64, linelength))
```

```
                goto error;


        DisposHandle (htext);
        IACreturntext (h64);
        return;
        error:


        IACreturnerror (1, "\perror encoding the Base 64 text");
        } /*base64encodeVerb*/


void base64decodeVerb (void) {
        Handle h64, htext;
        if (!IACgettextparam ((OSType) keyDirectObject, &h64))
                return;
        htext = NewHandle (0);
        if (!decodeHandle (h64, htext))
                goto error;
        DisposHandle (h64);
        IACreturntext (htext);
        return;
        error:
         IACreturnerror (1, "\perror decoding the Base 64 text");
        } /*base64decodeVerb*/
```

Well, that is all for now. Hope you liked the manual. Bye.

Ankit Fadia

ankit@bol.net.in

http://www.ankitfadia.com

To receive tutorials on EVERYTHING YOU DREAMT of written by Ankit Fadia in your Inbox, join his mailing list, by sending a blank email to: programmingforhackers-subscribe@egroups.com