

TECHNOTE: Understanding PackBits

By Cameron Birse
Revised by Michael Marinkovich
marink@applelink.apple.com
Apple Developer Technical Support (DTS)

This Technote describes the format of data packed by the Toolbox utility `PackBits` and documents a change to the `srcBytes` limit and to the possible worst case.

Although you can simply unpack this data using `UnPackBits`, Apple provides this information for the terminally curious and for those manipulating `PICT` files by hand.

See also the “Mathematical and Logical Utilities” chapter in *Inside Macintosh, Operating System Utilities*.

▲ **WARNING**

This format information is subject to change. ▲

Describing the Interface to the PackBits Routine

The “Mathematical and Logical Utilities” chapter of *Inside Macintosh, Operating System Utilities* describes the interface to the `PackBits` routine as follows:

```
PackBits(Ptr*srcPtr, Ptr*dstPtr, short srcBytes);
```

The accompanying text states that `srcBytes`, the length of your uncompressed data, should not be greater than 127, and that in the worst case, the compressed data can be `srcBytes + 1`. To pack more than 127 bytes, you had to break the data up into 127-byte groups and call `PackBits` on each group. Beginning with system software version 6.0.2, this limit of 127 bytes is no longer valid. The new limit is 32,767 bytes, which is the maximum positive number that `srcBytes` can hold. The worst case can be determined according to the following formula:

$$\text{MaxdestBytes} = (\text{srcBytes} + (\text{srcBytes} + 126) \text{ DIV } 127)$$

which is comparable to breaking up the data into 127-byte groups and picking up an additional byte for each group.

Specifying the Flag-counter Byte

The first byte is a flag-counter byte that specifies whether or not the following data is packed, and the number of bytes involved.

If this first byte is a negative number, the following data is packed and the number is a zero-based count of the number of times the data byte repeats when expanded. There is one data byte following the flag-counter byte in packed data; the byte after the data byte is the next flag-counter byte.

If the flag-counter byte is a positive number, then the following data is unpacked and the number is a zero-based count of the number of incompressible data bytes that follow. There are (flag-counter+1) data bytes following the flag-counter byte. The byte after the last data byte is the next flag-counter byte.

Given a pointer to the start of packed data, there is no way to know when you have reached the end of the packed data. Because `UnPackBits` requires the length of the unpacked data, you need to know either the length of the packed or unpacked data before you start unpacking.

Using PackBits

▲ WARNING

`PackBits` never generates the value -128 (\$80) as a flag-counter byte, but a few PackBits-like routines that are built into some applications do. `UnpackBits` handles this situation by skipping any flag-counter byte with this value and interpreting the next byte as the next flag-counter byte. If you're writing your own `UnpackBits`-like routine, make sure it handles this situation in the same way. ▲

Consider the following example:

Unpacked data:

```
AA AA AA 80 00 2A AA AA AA AA 80 00 2A 22 AA AA AA AA AA AA AA AA AA
```

After being packed by `PackBits`:

```
FE AA           ; (-(-2)+1) = 3 bytes of the pattern $AA
02 80 00 2A     ; (2)+1 = 3 bytes of discrete data
FD AA           ; (-(-3)+1) = 4 bytes of the pattern $AA
03 80 00 2A 22 ; (3)+1 = 4 bytes of discrete data
F7 AA           ; (-(-9)+1) = 10 bytes of the pattern $AA
```

or

```
FE AA 02 80 00 2A FD AA 03 80 00 2A 22 F7 AA
*   *           *   *           *
```

The bytes with the asterisk (*) under them are the flag-counter bytes. `PackBits` packs the data only when there are three or more consecutive bytes with the same data; otherwise it just copies the data byte for byte (and adds the count byte).

Note

The data associated with some PICT opcodes, \$0098 (PackBitsRect) and \$0099 (PackBitsRgn), contain PixData which is basically made of PackBits data. It should be noted, though, that the format for PixData includes a byteCount or length in addition to the data described in this Note. ♦

For example, the following is the result of decoding a sample PICT2:

```

data 'PICT' (25534) {
    0936 0000 0000 0007 001E          /* pic size, picFrame */
    0011 02FF                          /* pict2 */
    0C00                                /* header */
        FFFF FFFF 0000 0000 0000 0000 001E 0000 0007 0000 0000 0000
    001E                                /* def hilite */
    0001                                /* clipRgn */
        000A 0000 0000 0007 001E
    0098                                /* PackBitsRect */
        801E                                /* rowbytes of 30 */
        0000 0000 0007 001E                /* Bounds */
        0000                                /* packType */
        0000                                /* version */
        0000 0000                            /* packSize */
        0048 0000                            /* hRes */
        0048 0000                            /* vRes */
        0000                                /* pixelType */
        0008                                /* pixelSize */
        0001                                /* cmpCount */
        0008                                /* cmpSize */
        0000 0000                            /* planeBytes */
        0000 1F10                            /* pmTable */
        0000 0000                            /* pmReserved */
    /*color table*/
        0000 4CBC                                /* ctSeed */
        8000                                    /* ctFlags */
        00FF                                    /* ctSize */
            0000 FFFF FFFF FFFF
        ... /* 254 ColorSpec's omitted */
            0000 0000 0000 0000
    0000 0000 0007 001E                /* srcRect */
    0000 0000 0007 001E                /* dstRect */

```

TECHNOTE : Understanding PackBits

```
0000                                     /* srcCopy          */

/* Now we have the scan line data packed as follows:
[bytecount for current scan line] [data as defined above]
If rowBytes is > 250 then byteCount is a word else is a byte
(in this case, byteCount is a byte)
note that each unpacked row adds to 30 rowBytes
*/

/* line 1, byte count is 2 (best case for a row) */
02
    E3 FF          /* -(-29) + 1 = 30 FF's */
/* line 2, byte count is 19 (0x13) */
13
    01 FF 23      /* 1+1 data bytes      */
    FE 00          /* -(-2)+1 0's        */
    FC 23          /* -(-4)+1 0x23's     */
    FE 00          /* 3 0's              */
    FC 23          /* 5 0x23's           */
    FE 00          /* 3 0's              */
    FC 23          /* 5 0x23's           */
    FE 00          /* 3 0's              */
    00 FF          /* 1 data byte        */
/* line 3, byte count is 28 */
1C
    02 FF 00 23   /* 3 data bytes       */
    FE 00          /* 3 0's             */
    FE 23          /* 3 0x23's          */
    01 00 23      /* 2 data bytes       */
    FE 00          /* 3 0's             */
    FE 23          /* 3 0x23's          */
    01 00 23      /* 2 data bytes       */
    FE 00          /* 3 0's             */
    FE 23          /* 3 0x23's          */
    04 00 23 00 00 FF /* 5 data bytes     */
/* line 4, byte count is 31 (worst case for a row) */
1F
    03 FF 00 00 23 /* 4 data bytes      */
    FE 00          /* 3 0's             */
    00 23          /* 1 data byte       */
    FE 00          /* 3 0's             */
```

TECHNOTE : Understanding PackBits

```
    00 23          /* 1 data byte          */
    FE 00          /* 3 0's                                */
    00 23          /* 1 data byte          */
    FE 00          /* 3 0's                                */
    00 23          /* 1 data byte          */
    FE 00          /* 3 0's                                */
    00 23          /* 1 data byte          */
    FE 00          /* 3 0's                                */
    02 23 00 FF    /* 3 data bytes         */
/* line 5, byte count is 28 */
1C
    01 FF 00       /* 2 data bytes        */
    FE 23          /* 3 0x23's           */
    01 00 23       /* 2 data bytes        */
    FE 00          /* 3 0's              */
    FE 23          /* 3 0x23's           */
    01 00 23       /* 2 data bytes        */
    FE 00          /* 3 0's              */
    FE 23          /* 3 0x23's           */
    01 00 23       /* 2 data bytes        */
    FE 00          /* 3 0's              */
    FE 23          /* 3 0x23's           */
    00 FF          /* 1 data byte         */
/* line 6, byte count is 18 */
12
    00 FF          /* 1 data byte         */
    FC 23          /* 5 0x23's           */
    FE 00          /* 3 0's              */
    FC 23          /* 5 0x23's           */
    FE 00          /* 3 0's              */
    FC 23          /* 5 0x23's           */
    FE 00          /* 3 0's              */
    FD 23          /* 4 0x23's           */
    00 FF          /* 1 data byte         */
/* line 7, byte count is 2 (best case for a row) */
02
    E3 FF          /* 30 0xFF's          */
00 /* pad so next command starts at word boundary */

00FF          /*end of pic          */
};
```

Further Reference

- *Inside Macintosh, Operating System Utilities* – Mathematical and Logical Utilities
- Technical Note PT 24, “MacPaint Document Format”

Change History

This Technote was originally written in November, 1987.

Since November, 1990, a warning has been added about the handling of a flag-counter byte value of -128.

TECHNOTE : Understanding PackBits