**PackBits** Perform RLL byte compression on arbitrary data

#include <ToolUtils.h> **Toolbox Utilities**

```
void    PackBits(srcPtr, destPtr, srcLen);
Ptr     *srcPtr;     address of a pointer to data to pack
Ptr     *destPtr;    address of a pointer to a destination buffer
short   srcLen;      length of data to compress
```

**PackBits** compresses up to 32,768 bytes of data by replacing sequences of 3 or more identical bytes with a 2-byte code. It is most often used to compress image data, as found in a BitMap or a PixMap. The worst case compression can be calculated using the formula: (srcLen + (srcLen + 126) / 127).

*srcPtr* is the address of a pointer to some data to be compressed. Upon return, the pointer has been adjusted to just beyond the data that has been compressed; i.e., ready for the next call.

*destPtr* is the address of a pointer to a buffer to hold the compressed data. Upon return, it has been adjusted to just beyond the end of the compressed data.

*srcLen* is the size, in bytes, of the data to be compressed.

**Returns**: none

---

Notes: Since image data is often full of "white-space", this is ideal for packing data for long-term RAM storage and before writing it to disk or storing it in a resource. It is not quite so useful for compressing text or other data. Note the misnomer: **PackBits** does not pack bits; it packs sequences of bytes.

**PackBits** was formerly limited to compressing data in 127 bytes blocks. To compress more than 127 bytes, a programmer would have to break the data up and make multiple calls to **PackBits**. Starting with System Software version 6.0.2, this restriction has been lifted. If you want your program to run on Systems prior to version 6.0.2, you must obey the 127-byte limit.

Typical usage is to pack each line of a BitMap or PixMap separately, as in the example below. The example procedure, PackScreen, typically compresses a black and white screen image from 32K down to about 4K or 5K.

**Example**

```
#include <ToolUtils.h>

Ptr savePtr;
long    mapSize, PackScreen(Ptr);

mapSize = (long) screenBits.bounds.bottom * screenBits.rowBytes;
savePtr = NewPtr( mapSize + 1 );        /* allocate storage buffer */
```

```
mapSize = PackScreen( savePtr );
SetPtrSize( savePtr, mapSize );              /* shrink to actual size */


UnpackScreen( savePtr );
DisposPtr( savePtr );


/* ================== pack and save screen image to caller's buffer */
/* ================== returns size of packed screen */
long PackScreen(Ptr destBuf )
{
    short   j;
    Ptr srcPtr, destPtr;

    destPtr = destBuf;                       /* make a copy to use */
    srcPtr = screenBits.baseAddr;
    for ( j=0; j< screenBits.bounds.bottom; j++ ) {
        PackBits( &srcPtr, &destPtr, screenBits.rowBytes );
    }
    return( destPtr-destBuf );
}

/* ==================== unpack packed screen image, storing to screen */
void UnpackScreen(Ptr srcPtr )
{
    short   j;
    Ptr destPtr;

    destPtr = screenBits.baseAddr;
    for ( j=0; j< screenBits.bounds.bottom; j++ ) {
        UnpackBits( &srcPtr, &destPtr, screenBits.rowBytes );
    }
}
```