XBin Specifications

Introduction.

First of all, the default file extention of an XBin file is '.XB', just as a BIN file by default has '.BIN' as extention. Even though long filenames are getting popular due to newer operating systems like Windows 95, Windows NT, OS/2, Unix, Linux... there was specifically chosen for .XB over .XBin

An XBIN consists of 4 main parts, a header (required), a palette (optional), a font (optional), and the image data (optional).

Header (Required)
Palette (Optional)
Font (Optional)
Image Data (Optional)

Even though it may seem strange, an XBin is not **required** to have an image. You can use the standard XBin format for storing pre-made fonts and pre-made palettes which you could load from a viewer/editor (as alternate font/palette).

Header.

The XBIN header consists of 11 bytes. The header describes the size of the image, how the screen should be set up, and how the rest of the XBin should be processed.

Layout of XBin header

FieldName	Size	Type	Purpose
ID	4	Char	XBin identification, these 4 bytes should contain the text "XBIN" (All capitals). Any file which does not have a matching ID should not be considered to be an XBin file.
EOFChar	1	Char	End of file character (Ctrl-Z, Ascii 26, 1A hex) When a user uses the TYPE command to view the file, he'll just see "XBIN" printed on screen.
Width	2	Numeric	Width of the image in character columns.
Height	2	Numeric	Height of the image in character rows.
FontSize	1	Numeric	Fontsize 1 Numeric Number of pixel rows (scanlines) in the font, Default value for VGA is 16. Any value from 1 to 32 is technically possible on VGA. Any other values should be considered illegal.
Flags	1	Bits	A set of flags indicating special features in the XBin file. More on this later.

A sample XBin header in Pascal could be :

```
TYPE XB_Header = RECORD
                   ID      : ARRAY[0..3] OF Char;
                   EofChar : Byte;
                   Width   : Word;
                   Height  : Word;
                   FontSize: Byte;
                   Flags : Byte;
                 END;
```

A sample XBIN header in C could be :

```
typedef struct XB_Header {
   unsigned char   ID[4];
   unsigned char   EofChar;
   unsigned short  Width;
   unsigned short  Height;
   unsigned char   Fontsize;
   unsigned char   Flags;
};
```

Flags.

The 'flags' field in the XBin header tells us more on special features the XBin has in use. The flags field consists out of 8 separate bits each with it's unique function. Bits which are 0, are considered OFF or disabled, Bits with 1 are considered ON or enabled.

Flag byte bit usage.

7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Unused	Unused	Unused	512Chars	NonBlink	Compress	Font	Palette

Palette

Indicates if a palette is present (1) or whether no palette is present (0) and the default palette applies.

Font

Indicates if a font is present (1) or whether no font is present (0) and the default font applies. Since the VGA default fontsize is 16, this bit should be 1 for all other font sizes. In consequence, a font should then also be present in the XBin file.

Compress

Indicates if the XBin has compressed image data (1) or whether the image datais stored in raw memory format (0)

NonBlink

When 1, the image should be shown in non-blink (IceColor) mode.

512Chars

When 1, the image is built up out of 512 characters in stead of the usual 256. This bit also requires the **Font** bit to be set since the VGA has no default font for 512 character mode.

Palette.

A palette is only present when the '**Palette**' bit is set in the **Flags** field of the XBin header.

The palette is built up of 48 bytes, a red, green and blue value (in that order) for each of the 16 colors. Each palette value can range from 0 to 63.

Font.

A Font is only present when the '**Font**' bit is set in the **Flags** field of the XBin header.

For each character (256, or 512 when the '**512Chars**' bit is set in the **Flags** field) **FontSize** bytes are stored in sequence. The character set is defined from the top row of each character matrix to the bottom row.

In a 16 pixel high font, the first 16 bytes are the fontmatrix for ascii value 0, the next 16 are for ascii 1 and so on. In total, a 16 pixel font would have a font of 4096 bytes (16*256).

Technically, the biggest font possible would be 16Kb in size. (32 pixels high, 512 characters), and the smallest would be 256 bytes (1 pixel high, 256 characters).

Image data.

The image data is a raw image of video memory. Each character consists of 2 bytes, the first being the character, the second being the attribute (color).

The size of the image data would thus be equal to **Width*Height***2. The biggest XBin (not including header, palette and font) would be a whopping 8Gb (65535*65535*2) in size and the smallest would be 0 bytes (0*0*2).

Unless the '**Compress**' bit is set in the **Flags** field, image data is stored in the exact way you would need it in video memory.

When the '**Compress**' bit is set, image data is compressed with XBin-Compression. This is a fairly simple compression system, which should pose no real difficulty to decompress. As compression goes however an XBin compressor is a little harder to write.

XBin Compression

The XBin compression uses a slightly improved Run-Length Encoding scheme which will do very well on this type of data.

In stead of describing how a compressor would work, I'll explain how the compression works by giving some examples. In these examples, you'll see strings of characters more or less like this one :

```
Aa,Ab,Ac,Ba,Bb,Bc,De,Zx,Yu
```

This string represents a part of the uncompressed data. The capital letters are character bytes, the lower case letters are attribute bytes.

OK, pay close attention, as things may get hairy now ;-).

When you examine an ANSi or a BIN file, you may or may not have noticed several characteristics which are typical for ANSi/BIN files. You see sequences of identical characters one after another, and you see sequences of identical colors one after another. You may even see the combination of both; identical characters in identical colors one after another.

XBin-compression makes use of these characteristics by replacing sequences of identical characters/color with a counter and the actual data. A sequence like :

```
Aa,Aa,Aa,Aa,Aa,Aa,Aa,Aa,Aa,Bb,Bb,Bb,Bb
```

could easily be replaced with

```
[Repeat 9 times]Aa,[Repeat 4 times]Bb
```

The '`[Repeat x times]`' tag is the repeat counter.

If you're a smart observer, you've probably already figured where I'm driving at. It shouldn't be too hard to figure out that there's a need for four different types of compression :

1. No compression (when two subsequent character/attribute pairs have no relation).
2. Character compression (for a sequence of identical characters in different colors).
3. Attribute compression (for a sequence of different characters with identical attributes).
4. character/attribute compression (for a sequence of identical character/attributes).

Hmm... 4 types of compression.. That would nicely fit in 2 bits, leaving 6 bits in a byte unused. Now.. what if we were to use those 6 bits for the repeat counter... This is exactly what XBin compression does...

The XBin compression consists out of a sequence of repeat counters followed by the appropriate number of data bytes.

Before we go any further now, allow me to make an important note. XBin compression works on a **ROW** by **ROW** basis. The compression does **NOT** carry through to the next line. So, if you would have for example two lines like this :

```
Ab,Aa,Aa,Aa
Aa,Aa,Aa,Ab
```

It should **NOT** be encoded as

```
Ab,[Repeat 6 Times]Aa,Ab
```

but **SHOULD** be encoded as

```
Ab,[Repeat 3 Times]Aa
```

```
[Repeat 3 Times]Aa,Ab
```

The are several reasons why it should work like this :

- on-the-fly decoding is facilitated (more on this later)
- You could run into a problem with odd XBin widths since video memory memory always has an even width. You'd have to implement special code to skip over this one odd byte.
- You at least have SOME way of detecting errors in the XBin file. Whenever a line doesn't nicely work out to the required width there must be some error in the XBin file (or a bug in your decoding routine ;-))

XBIN Compression continued.

The repeat counter byte is split up in two parts, the two most significant bits are the compression type, the six least significant bits are the actual repeat counter.

Repeat-counter layout

7	6	5	4	3	2	1	0
Compression Type			Repeat counter				

0	0	No compression
0	1	Character compression
1	0	Attribute compression
1	1	Character/Attribute compression

Since a repeat of 0 is downright useless, we'll store the repeat counter as one less of it's actual number of repeats.

A value of 0 indicates a repeat of 1.
A value of 1 indicates a repeat of 2.
And so on, goving a us a maximum of a 64-repeat.

In the examples, the Repeat counter byte will be used as `[Type,Count]` Where **`Type`** is `00`, `01`, `10` or `11`, and **`Count`** is a number from 0 to 63.

For example, [01,10] would mean, Attribute compression with a repeat counter of 10 (11 effective repeats).

Compression type 00 : No compression.

This type of compression is needed whenever two or more sequences of character/attribute pairs have nothing in common. The repeat counter is followed by the appropriate number of character/attribute pairs.

Data :

```
AaBbCcDdEeFfGg
```

XBin-Compressed :

```
[00,6]AaBbCcDdEeFfGg
```

In a worst-case situation where you would have to use **No compression** on the entire file, this would mean the 'compressed' data is bigger than the non-compressed data. In this situation, the

best thing to do would be to store the image data uncompressed, and set the '**Compress**' bit in the **Flags** field to 0.

Compression type 01 : Character compression.

This type of compression is used whenever a sequence of identical characters is found, but where the attribute changes. The repeat counter is followed by the character to use which is in turn followed by the appropriate number of attribute bytes.

Data :

```
AaAbAcAdAeAfAg
```

XBin-Compressed :

```
[01,6]Aabcdefg
```

Compression type 10 : Attribute compression.

This type of compression is used whenever a sequence of identical attributes is found, but where the character changes. The repeat counter is followed by the attribute to use which is in turn followed by the appropriate number of character bytes.

Data :

```
AaBaCaDaEaFaGa
```

XBin-Compressed :

```
[10,6]aABCDEFG
```

! Note this is the only time you'll see the attribute byte BEFORE the characters.

Compression type 11 : Character/Attribute compression.

This type of compression is used whenever a sequence of identical character attribute pairs is found. The repeat counter is followed by the character attribute pair.

Data :

```
AaAaAaAaAaAaAa
```

XBIN Compressed :

```
[11,6]Aa
```

You'll find example code on how to do XBin compression and decompression in the XBin Developers Kit. Decompression is also explained in the XBin Tutorial.

Back to XBIN

This page is being maintained by

Tasmaniac@acid.org