0018

COLLABORATORS

	TITLE : 0018		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 8, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

0018

1.1 Personal Fonts Maker - O. PFM IFF Data Format

O.   PFM IFF Data Format

As explained in sections 4.3 ("Load PFM Font"), 4.4 ("Save PFM Font"), 4.8 ("Load Character Set") and 4.9 ("Save Character Set"), the Personal Fonts Maker uses different IFF conforming formats to load and save its font and character set files. This appendix contains the necessary technical documentation to create external programs that can access and create files which are compatible with the Personal Fonts Maker formats. Programmers are encouraged to use this information to write their own routines to handle fonts and character sets.

This appendix assumes that the Commodore "'EA IFF 85' Standard for Interchange Format Files" documentation has been understood. A disk for software developers, which includes the "ParseCPFM" tool to analyze font files and character set files, is available from Cloanto.

O.1   The IFF FORM: CPFM

A CPFM (Cloanto Personal Fonts Maker) IFF FORM contains one or more IFHD (InFormation HeaDer) chunks, each (not necessarily immediately) followed by a CHDT (CHaracter DaTa) chunk. Each IFHD describes the contents of the CHDT chunk which follows. The optional CSNM (Character Set NaMe) chunk contains the name of the character set used to create the data described by the preceding chunks. Reference point positions are stored in the optional REFP (REFerence Points) chunk. Additional chunks may be added in future versions of the program to support new functions.

An IFHD chunk gives some general information on the following CHDT chunk. The data contained in the IFHD chunk tells the program whether the CHDT which follows contains data relative to a font or to a character set. Information like the maximum size of the character images, the aspect ratio (horizontal and vertical densities) and other data is also stored in the IFHD chunk. The CHDT chunk contains the data relative to each character, like the graphical image and the position in the encoding vector (if the chunk contains character set data) of each character. The IFHD chunk is called a property chunk, and must appear before any CHDT chunk. The CSNM chunk must come after (not necessarily immediately after) the IFHD chunk it refers to.

A character set needs to store information which is very similar to that of a font. The default images of the characters are treated similarly to the images of the characters in a font. Additional data regarding the character encoding vector is also contained in the character set data. A couple of IFHD+CHDT chunks either contain font data or character set data. It is possible, even if not yet supported by the program, that the same file may contain information regarding a character set (a couple of chunks), and a font designed having that character set in mind (other two chunks).

O.2 IFF Chunks: IFHD

An IFHD (InFormation HeaDer) chunk contains an instance of an InformationHeader structure:

```
struct InformationHeader
{
 UWORD MaxWidth, MaxHeight;  /* maximum character size */
 UWORD HorizDPI, VertDPI;    /* print/display density */
 UWORD MaxBytesPerLine;      /* bytes sufficient to store an image row */
 UBYTE BitPlanes;            /* number of bitplanes */
 UBYTE System;               /* file creator’s operating system */
 ULONG Flags;                /* font or character set flags */
};

/* System */
#define IHS_AMIGA_SYSTEM       0
#define IHS_MSDOS_SYSTEM       1

/* Flags */
#define IHF_ATTRIBUTES          0x0000FFFF /* mask for font/cset attr. */
#define IHF_ITALIC              0x00000001
#define IHF_BOLD                0x00000002 /* excludes IHF_LIGHT */
#define IHF_LIGHT               0x00000004 /* excludes IHF_BOLD */
#define IHF_UNDERLINE           0x00000008
#define IHF_OUTLINE             0x00000010
#define IHF_SHADOW              0x00000020
#define IHF_SUPERSCRIPT         0x00000040 /* excludes IHF_SUBSCRIPT */
#define IHF_SUBSCRIPT           0x00000080 /* excludes IHF_SUPERSCRIPT */
#define IHF_ENLARGED            0x00000100 /* excludes IHF_CONDENSED */
#define IHF_CONDENSED           0x00000200 /* excludes IHF_ENLARGED */
#define IHF_REVERSE             0x00000400
#define IHF_SERIF               0x00000800
#define IHF_DRAFT               0x00001000
#define IHF_FIXED_PITCH         0x00002000
#define IHF_RIGHT_TO_LEFT       0x00004000
#define IHF_LANDSCAPE           0x00008000

#define IHF_FONT_HEADER         0x80000000 /* font IFHD+CHDT (no cset) */
```

Fields are filed in the order shown. The compiler must not add pad bytes to the structure. UWORD and ULONG are high bytes first.

Values in MaxWidth and MaxHeight must be in the range 1-65535. The two values must be 16 and 18 (decimal), respectively, if the IFHD refers to a character set (rather than a font).

Valid values for HorizDPI and VertDPI range from 0 to 65535. If one of the two is 0, the program uses its own current aspect ratio. The two fields are ignored (the screen display ratio is used) if the data is read as a character set, but should always be filled in correctly when writing the character set data. The ratio of (a) equals the ratio of (b) when XDPI(a) / YDPI(a) == XDPI(b) / YDPI(b). This is important to understand when and how images are stretched.

MaxBytesPerLine is MaxWidth divided by 8. The value must be 2 if it refers to character set data (rather than font data).

The Personal Fonts Maker does not read the encoding vector of character set data if the System field is not set to the same operating system under which the program is running. For example, the Amiga version of the Personal Fonts Maker reads only the image data in a character set's CHDT chunk if the operating system field is not set to IHS_AMIGA_SYSTEM.

The Flags field should have the IHF_FONT_HEADER flag set if the chunk refers to font data, and cleared (not set) if a character set is being described instead. In both cases (description of a font or character set), different font attribute flags can be used, keeping in mind that some attributes mutually exclude each other (e.g. IHF_SUPERSCRIPT and IHF_SUBSCRIPT cannot be set at the same time).

The ATRB Font Format Description Language variable (section 2.7.2.1) can be extracted from the Flags field using the following formula:

```
  ATRB = InformationHeader.Flags & IHF_ATTRIBUTES
```

BitPlanes must contain a value in the range 0 to 255. A value of 0 means that all characters are empty.

Programs reading the IFHD chunk must be able to support future extensions of the data format. The size of the InformationHeader structure can increase. Programs reading this data should seek from the end of the structure which was read, to the end of the chunk. The number of bytes to skip can be calculated with a similar formula:

```
  SkipBytes = IFHDChunkLength - sizeof(struct InformationHeader)
```

O.3 IFF Chunks: CHDT

The CHDT (CHaracter DaTa) chunk contains an ordered sequence of character units, each referring to a character in the font or character set being described. The units are divided into three parts: the units's format descriptor, a character header, containing non-graphical data, and a variable-length part containing the data associated with the character image of the character referenced to.

The format descriptor byte specifies the format of the character header and the Cloanto Character Image Format (CCIF) used to achieve the required level of data compression and access speed. Using the CCIF compression algorithms described here, the font files on the Personal Fonts Maker disks have been reduced by an average of 47% of their uncompressed size, and the character sets by more than 55%.

The image data is divided into bitplanes, which in turn are divided into horizontal rows, each described by a maximum of MaxBytesPerLine bytes. Compressed graphical data is stored bytewise (8-bit bytes) or in nibbles (4-bits), depending on the compression technique. If groups of identical bits are not compressed, the data is stored bitwise.

Programs handling CCIF data must always read and write across rows and bitplanes. All rows within the specified frame, of all bitplanes, are merged into one ideal row of bits, left to right, top to bottom, from one bitplane to the next.

No pad bytes are used between the character units. Adjacent character units must not necessarily refer to contiguous characters, but they must be ordered by the position of the character in the font, from the beginning to the end (i.e. lower character codes first), though the Personal Fonts Maker can load character units even if they are in "sparse" order. A program loading character units from a CHDT chunk can determine when the last unit was read by verifying when the end of the chunk is reached. As specified in the IFF documentation, odd-length chunks must be followed by a pad byte.

The following data structures are used:

```
UBYTE FormatDescriptor;

struct FontCImageHead8
{
 UBYTE CharNum;         /* position of the character in the font */
 UBYTE XSize;           /* horizontal image size in dot-columns (XSIZ) */
 BYTE  Space;           /* total space before next character (SPCE) */
 BYTE  Offset;          /* offset of this character's beginning (KERN) */
};

struct FontCImageHead16
{
 UWORD CharNum;         /* position of the character in the font */
 UWORD XSize;           /* horizontal image size in dot-columns (XSIZ) */
 WORD  Space;           /* total space before next character (SPCE) */
 WORD  Offset;          /* offset of this character's beginning (KERN) */
};

struct CSetCImageHead8
{
 BYTE CharNum;               /* position in the character set */
 BYTE EqualToSystemChar;     /* code in the host system's character set */
};

struct CSetCImageHead16
{
 WORD CharNum;               /* position in the character set */
 WORD EqualToSystemChar;     /* code in the host system's character set */
};

/* EqualToSystemChar */
```

```
#define NOT_IN_SYSTEM_SET       -1     /* no equivalent in system cset */

struct CCIFPlaneInfo
{
 UBYTE PlanePick;            /* planes described by graphical data */
 UBYTE PlaneOnOff;           /* status of other planes */
};

struct CCIFFrame8
{
 UBYTE BlankColumns;
 UBYTE BlankRows;
 UBYTE DataColumns;
 UBYTE DataRows;
};

struct CCIFFrame16
{
 UWORD BlankColumns;
 UWORD BlankRows;
 UWORD DataColumns;
 UWORD DataRows;
};

/* FormatDescriptor */

#define FD_CIHEAD8      0x01
#define FD_PLANEINFO    0x02
#define FD_FRAME8       0x04
#define FD_FRAME16      0x08
#define FD_PACKET4      0x10
#define FD_PACKET8      0x20
#define FD_RESERVED     (0x40 | 0x80)
```

The FormatDescriptor byte describes exactly which structures follow in the data unit, and how the graphical data is stored. The eight bits in this byte store the following information:

```
  FD_CIHEAD8  : compact (8-bit) CSet/Font-CImageHead structure follows if
                the bit is set. Default is normal (full) structure.

  FD_PLANEINFO: specifies whether bitplane information data follows.

  FD_FRAME8   : CCIF 8-bit frame data structure follows.

  FD_FRAME16  : CCIF 16-bit frame data structure follows. Setting both
                FD_FRAME8 and FD_FRAME16 bits is a reserved condition,
                reserved for future implementations.

  FD_PACKET4  : CCIF "1+3"-bit packets are used.

  FD_PACKET8  : CCIF "1+7"-bit packets are used. Setting both FD_PACKET4
                and FD_PACKET8 bits is currently forbidden and remains
                reserved for future implementations. It does not make
                sense to set FD_FRAME8, FD_FRAME16, FD_PACKET4 or
                FD_PACKET8 if no image data follows (zero
                character-width).
```

FD_RESERVED : reserved for future versions, must be set to 0.

It is currently not permitted to set both FD_FRAME8 and FD_FRAME16 bits, or both FD_PACKET4 and FD_PACKET8 bits, or any of the FD_UNUSED bits.

Depending on whether a font or a character set is being described, either the FontCImageHead or the CSetCImageHead structure is used to introduce each character unit. An instance of the FontCImageHead or CSetCImageHead structure must exist in each character unit. If the FD_CIMHEAD8 bit of the format descriptor byte is set, the compact version of these structures is used. The compact structures occupy half the space of their full-length counterpart. As a trade-off, the compact structures can only be used if each of the values to be stored in the fields of the structure can be represented using 8 bits. This means that all unsigned values must be in the range 0-255 and signed values must be in the range from -128 to 127. It is important not to forget that even the CharNum field can exceed this limit, if it refers to the undefined characer (decimal code 256).

The FD_PLANEINFO bit of the format description byte specifies whether an instance of a CCIFPlaneInfo structure follows. Such a structure is useful when coloured fonts are processed, or to compress those characters which are completely empty (all dots in the background colour) or consist of one filled rectangle. Multiple-colour images are stored in more than one bitplane. For a given character one or more bitplanes may contain either all ’0’ or all ’1’ bits. In this case, memory can be saved if a more compact description of these bitplanes is saved, rather than the bitplanes themselves. If no memory can be saved in this way, the second bit of the format description byte should be cleared. The bits in the PlanePick field of the CCIFPlaneInfo structure tell the program which planes follow in the character image data. The status of the PlaneOnOff bits determines how the remaining bitplanes are to be set (all ’0’s or all ’1’s). Only the dots included in the frame described by the CCIFFrame structure (if present) are affected. Even if the BitPlanes field in the InformationHeader structure contains a value higher than 8, only the first eight bitplanes can be picked, cleared or set with this method. Any bitplanes after the first 8 are considered to be "picked".

One of the simplest measures that can be taken in order to store a character’s image data in the least possible space is to "cut" away the outermost empty rows and columns of the image. The first two fields in the CCIFFrame respectively specify how many blank columns and rows precede the region of coloured dots, starting from the top-left corner. These blank dots do not need to be stored in the image data. The other two fields indicate the number of columns described by the following image data, and how many image rows are described. The image data therefore only describes the content of the smallest possible rectangle containing all coloured dots. If the dimensions of this rectangle do not exceed 256 dots, and the top and left offset of the rectangle do not exceed this value either, the more compact 8-bit fields can be used. The bits FD_FRAME8 and FD_FRAME16 of the format descriptor byte specify which structure to use. If neither of the two bits is set, all dots which make up the character image must be stored.

Certain limit-cases are acceptable, but should be avoided. For example, if there are no outer empty rows or columns, it is a waste to use the CCIFFrame structure. This is also true if there are no dots in the character (zero-width character, not stored by the Personal Fonts Maker).

The bits FD_PACKET4 and FD_PACKET8 of the format descriptor byte specify whether the rows of dots which make up the character image are packed into nibbles or bytes. If neither of the two bits is set, the data is stored bitwise. Images are processed left to right, top to bottom, bitplane by bitplane. The leftmost dots are stored in the most significant bits in each byte. After all bits have been written, any remaining bits in the current byte are set to zero and used as pad bits.

If the FD_PACKET4 bit is set, the content of each row of bits in a bitplane is compressed into a sequence of packets. The packets are 4-bits long. The most significant bit in a packet determines the status ('0' or '1') of the data bits which are described. The remaining three bits contain a number from 1 to 8 (normally this would be 0 to 7, but 1 is always added since it does not make sense to describe zero bits) indicating how many bits have the specified value (i.e. 0 or 1). If there are more than 8 bits with the same value, more packets with the same setting of the most significant bit can be generated. This technique is very efficient to compress the description of rows of dots in a font's character image, especially if it is combined with a CCIFFrame structure. If the last packet in the last row of the last bitplane terminates in the middle of a byte, the remaining four bits are set to zero and used as pad bits.

If the FD_PACKET8 bit of the format descriptor byte is set, 8-bit packets are used. Up to 128 bits of data can be compressed into such a packet. This can be more space-efficient in extremely large or very high resolution fonts, and to describe long sequences of identical bits across rows of smaller characters.

Once again, it should be noted that if a CCIFFrame structure is used, only the rows which are contained in the specified frame are affected by the CCIFPlaneInfo structure and are described by the 4 or 8 bit packets, or by the bitmap data.

The order in which the different data structures are to be written in the CHDT chunk is the following (optional structures are enclosed by parentheses):

```
  FormatDescriptor
  FontCImageHead8 | FontCImageHead16 | CSetCImageHead8 | CSetCImageHead16
  (CCIFPlaneInfo)
  (CCIFFrame8 | CCIFFrame16)
  (graphical data)
```

It is reasonable not to write any graphical data if a character consists of zero rows and/or columns (it is however preferable not to save such characters at all), or if the CCIFPlaneInfo structure (perhaps combined with CCIFFrame) has already given a complete description of the character.

The data compression techniques supported by the Cloanto Character Image Format are extremely easy to handle. Fonts compressed in this way can fit into as little as 40 percent of the space occupied by the same non-compressed font. It is up to the programs writing the character units to choose the most appropriate format to store each character's data. Different techniques can be combined. A compression method should never be applied to a character if the resulting data would be longer than the non-compressed data. This is possible, especially using very small font formats (e.g. 6 x 5 characters).

O.4 IFF Chunks: CSNM

The CSNM chunk contains the name of a character set. This is the name of the character set which was used when the font was designed, if the chunk's data regards a font (rather than a character set). The CSNM chunk may be omitted if the character set is unknown (e.g. if no character set was selected). If the CSNM chunk follows an IFHD chunk describing character set data, the string contains the name which was originally assigned to the character set by the user (in the save command), i.e. the original file name (without path). This enables the program to detect whether the file has been renamed, and to display a warning message. It is better not to rename character set files, as they may be referenced by fonts created using these sets. Sections 4.3 ("Load PFM Font") and 4.9 ("Save Character Set") explain in more detail what the Personal Fonts Maker uses this name for.

The CSNM chunk may appear only after the IFHD chunk describing the data to which CSNM refers. The CSNM chunk may appear either before or after the CHDT chunk. A CSNM chunk does not refer to the whole CPFM FORM, but is always relative to the current couple of IFHD and CHDT chunks. The use of the CSNM chunk is optional.

The string contained in CSNM is an exact copy of the character set file name. The same character set and letter-case (capitals vs. lower case) used for the file name are used. Characters whose code is out of the range 33-126 should be avoided, as these may have different meanings on operating systems other than the one where the file was originally written. The maximum length for the file name is 63 characters (without zero terminator). At read time, the length of the string is determined by the chunk length. The string is not zero-terminated.

O.5 IFF Chunks: REFP

This (optional) chunk contains the vertical positions of one or more reference points. The REFP chunk must always contain data for at least four reference points, describing the following lines (from the top): cap line, mean line, baseline and underline position. These values must always be the first four in the REFP chunk. Section 3.21 ("Reference Points") has more on this subject.

The Personal Fonts Maker supports a minimum of four reference points. Other programs may support a minimum of more than four reference points. If the REFP chunk being read contains fewer values than the minimum number of reference points used by the program which is reading the data, the reference points to which no value could be assigned should be reset to a default value.

If the REFP chunk contains more values than the maximum number of reference points supported by the reader program, only the first values should be read. The values are always ordered, from the first reference point to the last. The first reference point is always reference point number one. Each value, ranging from 0 to YMAX, is stored as an unsigned word (high byte first). Thus:

```
  NumRefP = REFPChunkLength / 2
  ReadBytes = (NumRefP > MaxRefP ? MaxRefP : NumRefP) * 2
  SkipBytes = REFPChunkLength - ReadBytes
```

The values can range from 0 to MaxHeight (IFHD chunk), or from 0 to 18 if IFHD and REFP refer to a character set.