Cal Poly 4/2007

Document# $27\pi e^4$

Apple Lisa Information

FILE NAME

Apple Lisa Font Format

DISK #

COMMENTS

16 April 2007
Rebecca Bettencourt

16 pages

David T. Craig
736 Edgewater, Wichita, Kansas 67230
(316) 733-0914

Lisa

Apple Lisa Font Format

The Lisa Office System's fonts are stored in two files, `font.heur` and `font.lib`. This document describes the format of these files.

Endian Note: All numeric fields are big endian, as the 68000 processor used in the Lisa was big endian.

Note: Since this had to be reverse-engineered, I can guarantee that it is rife with inaccurate information. Wherever I took a complete guess at something, I have put a question mark. The other information should be considered an incomplete guess.

font.heur

The `font.heur` file is used to locate fonts based on a unique ID number. This would be the ID number passed to QuickDraw.

Note: If you are using `lisafsh-tool` to extract the `font.heur` file, the first 224 bytes of the file will be split off into a separate `font.heur.meta` file. The contents of the `font.heur.meta` file must be put back at the beginning of the `font.heur` file.

The file starts with a single byte indicating the number of fonts described. Following the first byte is a record for every font.

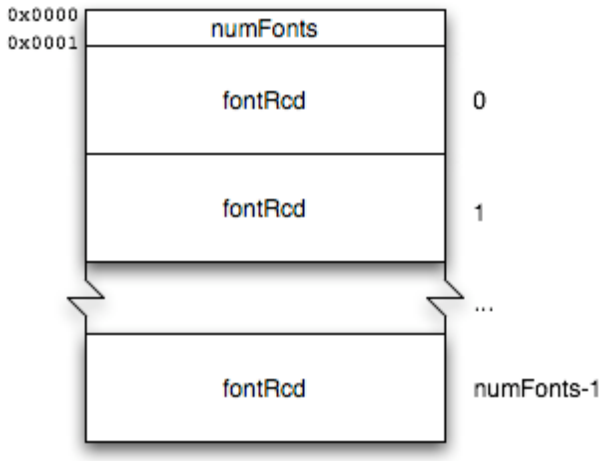

Figure 1 - Format of font.heur

Every record starts with a header of 10 bytes. Byte 0 is the font ID. Following the header is yet another set of records, each containing the name of the font and some other fields. I suspect these records are for different devices, though I am not sure. Byte 9 indicates the number of these records. The purpose of the other fields is not yet known.

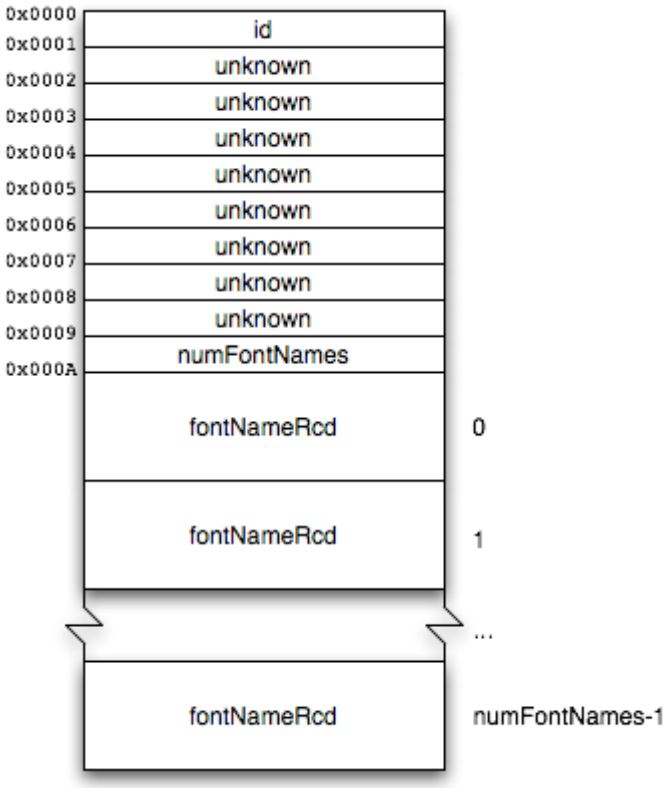

Figure 2 - Format of fontRcd

Each record inside a font record contains an 8-byte header and the name of the font as a Pascal string. There is no word boundary alignment, which suggests these unknown fields are byte values, not word values. Usually every record contains the same font name.

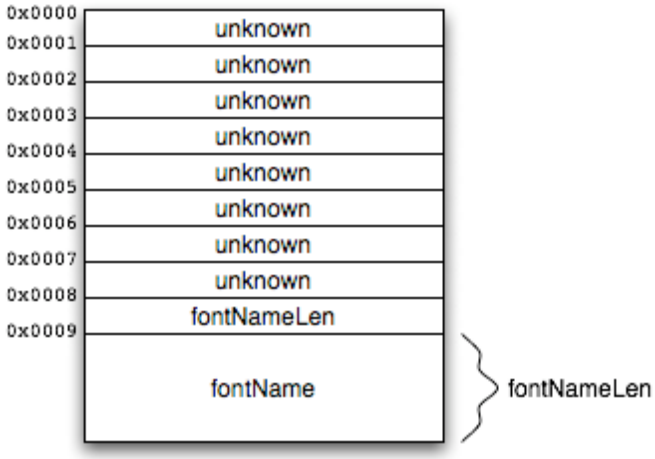

Figure 3 - Format of fontNameRcd

The known font names are listed below.

Font Name	ID #	Description
APPLE *		Apple daisy-wheel font
CALCFONT	18	Calculator buttons
CENTURY12R12	13	Classic 10-point 12-pitch
CENTURY12R10	14	Classic 12-point 10-pitch
CENTURY12RP	10	Classic 12-point
CENTURY14	16	Classic 14-point
CENTURY18RP	11	Classic 18-point
CENTURY24RP	12	Classic 24-point
HELV7RP	21	Desktop icon name font
HELV14	15	Modern 14-point
HELV18	5	Modern 18-point
HELV24	6	Modern 24-point
ICONFONT	22	Large desktop icons
IMGRAPH	24	LisaGuide graphics?
MARKER	20	LisaGraph data point markers
MFI *		Daisy-wheel font
MFR *		Daisy-wheel font
MPF *		Daisy-wheel font
MPP *		Daisy-wheel font
QUMEF *		Qume daisy-wheel font
QUMEP *		Qume daisy-wheel font
SYSCURSOR	3	Cursors
SYSPAT	2	LisaDraw patterns
SYSTEXT	0	System text font
TILE7R20	19, 27	8-point 20-pitch (Classic and Modern)
TILE7R15	7	8-point 15-pitch (Classic and Modern)
TILE12R12	8, 26	Modern 10-point 12-pitch
TILE12R10	9	Modern 12-point 10-pitch
TILE12RP	4	Modern 12-point
TILE20VT	23	20-pitch (132-column) VT box drawing characters
TILE12VT	17	12-pitch (80-column) VT box drawing characters
TOOLKIT	25	GUI elements
WMFONT	1	GUI elements and small desktop icons (Window Manager font)

* These fonts are used for daisy-wheel printers. They cannot be used by QuickDraw and do not have ID numbers, so they will not appear in `font.heur`. They will, however, appear in `font.lib` when we get to that later.

Table 1 - Known Lisa font names

font.lib

The `font.lib` file is far more interesting, as it contains the actual fonts themselves.

Note: If you are using `lisafsh-tool`, this file may be reported as `font.libf`. I believe this is due to bugs in the Lisa file system code. Also, when you extract it, the first 224 bytes of the file will be split off into a separate `font.lib[f].meta` file. The contents of the `font.lib[f].meta` file must be put back at the beginning of the `font.lib[f]` file.

The file starts with a header, enumerating the font resources contained inside and where in the rest of the file they can be found. The first word is the number of words (not bytes!) from where the font resource records start at `0x0004` to the end of the header.

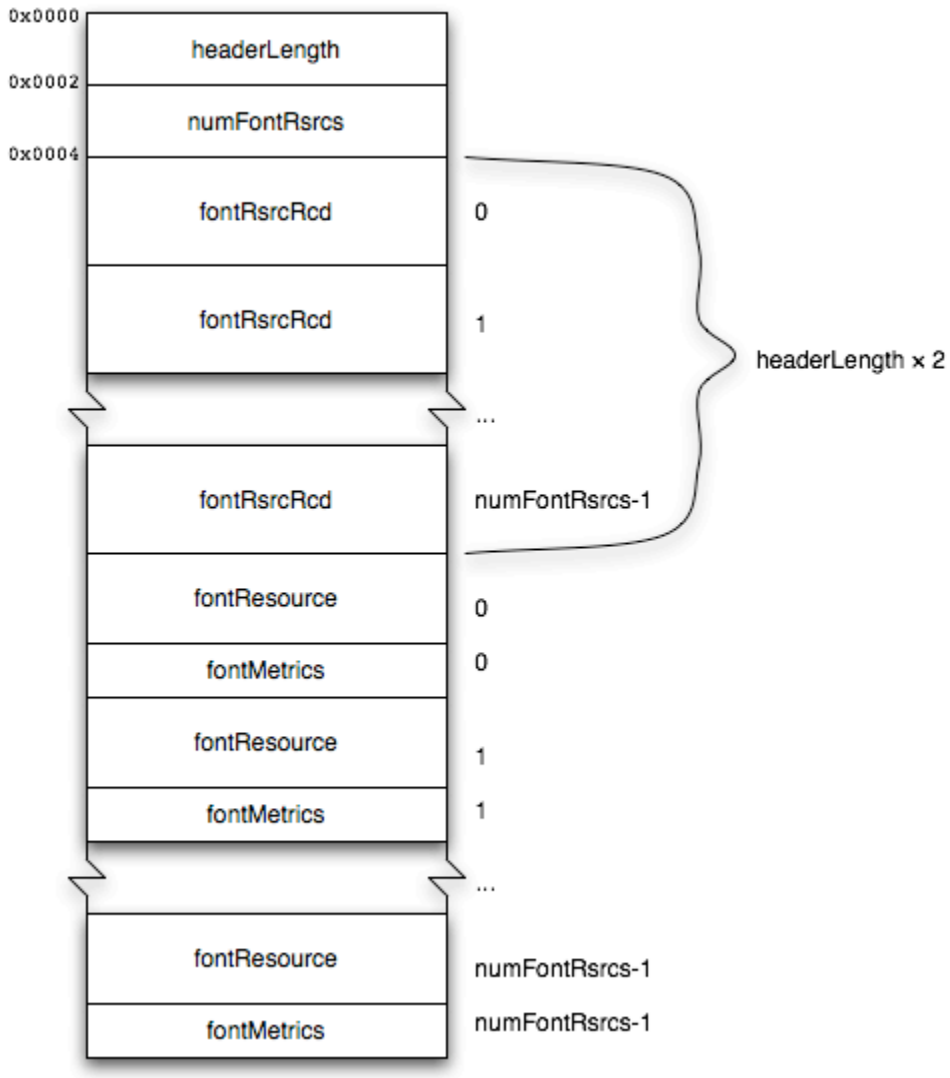

Figure 4 - Format of font.lib

Each record in the header contains the font name as a Pascal string and the offsets from the end of the header to the font information. Word boundary alignment is used, so the offsets always fall at even addresses. If the font name contains an even number of characters (therefore an odd number of bytes including the length byte), an extra byte is added. The offsets are long words, and expressed in terms of number of words (not bytes!) from the end of the header.

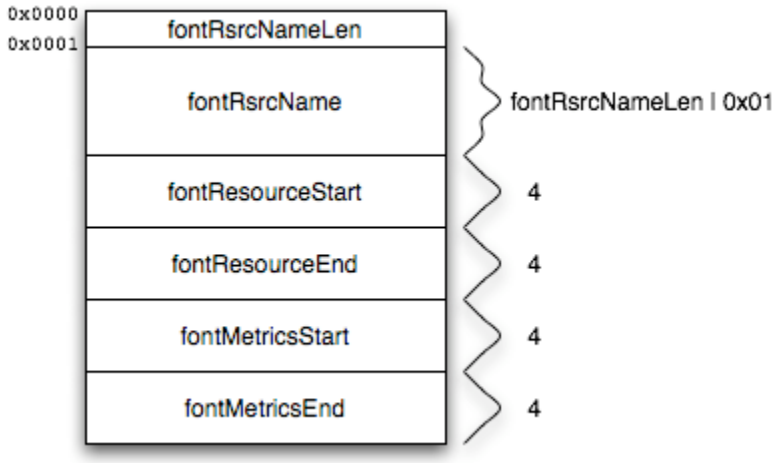

Figure 5 - Format of fontRsrcRcd

The font names in `font.lib` have a bit of extra information attached to them. After the name as listed in Table 1, there is a single character to indicate which device the font was created for, followed by a file extension of `.F`. For example, the `APPLE` font made for daisy-wheel printers is listed in `font.lib` as `APPLE`**`W.F`**. The Modern 18-point font made for laser printers is listed as `HELV18RP`**`T.F`**. The system text font made for the screen is listed as `SYSTEXT`**`S.F`**.

Character	Device
`W`	Daisy-wheel printer
`S`	Screen
`C`	Dot-matrix printer?
`H`	High-resolution dot-matrix printer?
`T`	Laser printer?

Table 2 - Device type suffixes for font names in font.lib

Font metrics are simply truncated Mac OS `FONT`/`NFNT` resources, containing no bitmap and only an offsets and widths table.

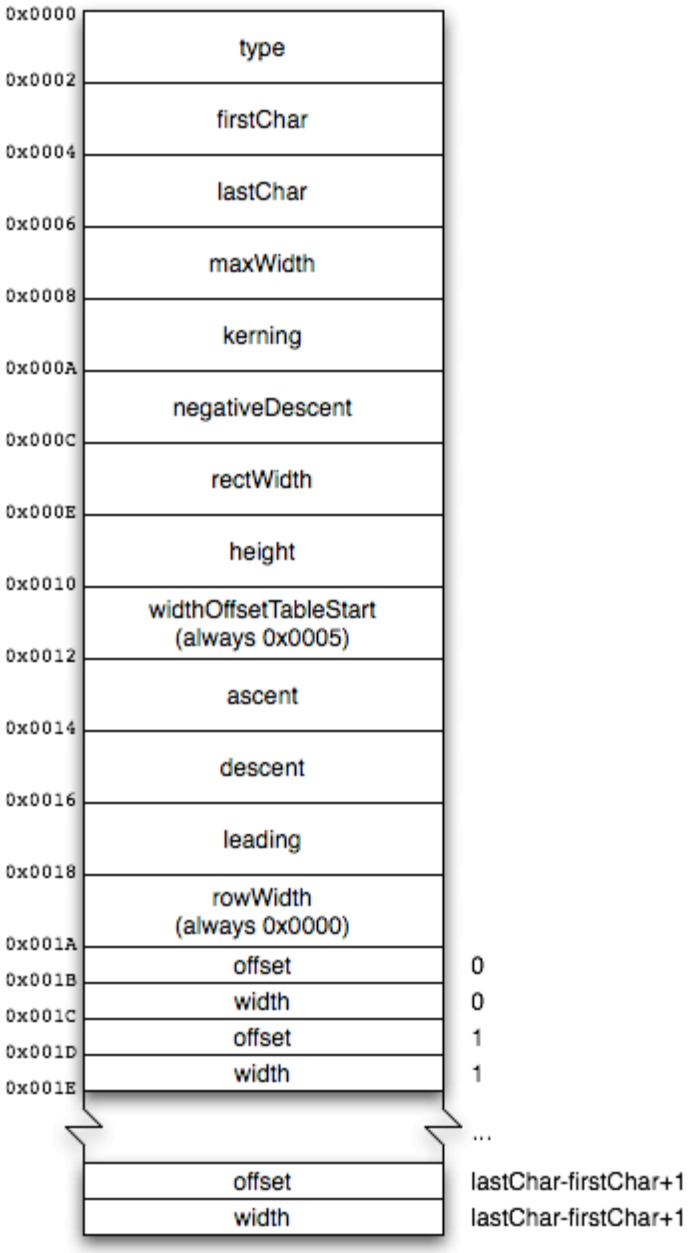

Figure 6 - Format of fontMetrics

Since the offsets and widths table starts immediately after the header, the field at `0x0010` is always `0x0005` (the table is five words after `0x0010`). Since there is no bitmap, the row width is zero. There are (lastChar-firstChar+2) entries in the offsets and widths table, because the last entry is for the catchall character (the one displayed when the actual character is undefined in the font; typically a white question mark in a black box). The width is the number of columns of pixels the character occupies. The offset is the number of columns of pixels to the left of the character that are completely white. If a character in the middle of the range of defined characters is undefined, its width and offset are both -1 or `0xFF`.

The format of the font resource, which contains the actual letterforms, varies somewhat. Every kind begins with a type.

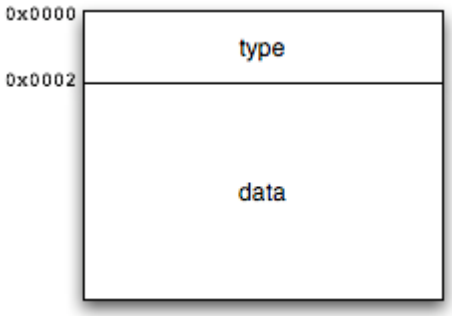

Figure 7 - Generic format of fontResource

The currently known types are listed below.

Type	Description
`0x0005`	Daisy wheel font (uncompressed)
`0x0006`	Laser font? (uncompressed)
`0x0086`	Laser font? (compressed)
`0x9000, 0xB000`	Bitmap font (uncompressed)
`0x9080, 0xB080`	Bitmap font (compressed)

Table 3 - Font resource types

This document will only describe the format of types `0x9000`, `0x9080`, `0xB000`, and `0xB080`.

Compressed Bitmap Fonts

The bitmap fonts in `font.lib` are compressed to save disk space. Following the type are the length of the compressed data, the length of the uncompressed data, and the compressed data itself.

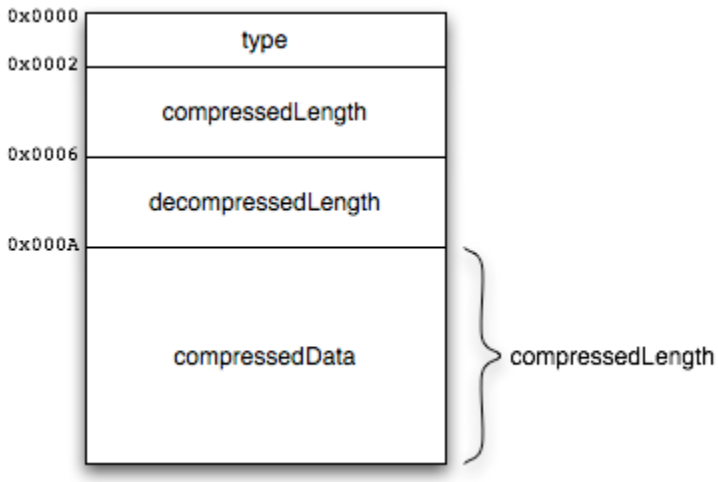

Figure 8 - Compressed bitmap font format

The decompression algorithm is simple:

- Create a buffer of the appropriate uncompressed length, rounding up to the nearest multiple of 8.
- Copy the compressed data to the start of the buffer.
- Starting at the end of the compressed data and the end of the buffer:
    - Get a byte from the compressed data (and move back a byte)
    - For each bit in that byte, starting with the least significant:
        - If the bit is set, write a zero byte to the buffer (and move back a byte)
        - If the bit is cleared:
            - Get a byte from the compressed data (and move back a byte)
            - Write that byte to the buffer (and move back a byte)
- Stop when you've reached the start of the buffer.
- For each row of the font bitmap, starting with row 1 (the second row):
    - XOR that row with the previous row

This is almost the same as the Lisa ROM's algorithm for decompressing icons, only with the decompression performed in reverse and the XOR applied only to the font bitmap. When the bitmap font is fully decompressed, it is exactly the same as a Mac OS `FONT`/`NFNT` resource.

Uncompressed Bitmap Fonts, a.k.a. `FONT`/`NFNT` Resources

This information is documented in *Inside Macintosh: Text*, but I have repeated it here for the sake of completeness.

An uncompressed bitmap font starts with a header of 26 bytes, including the type field. Following the header is a bitmap image, with all the glyphs in the font strung together with no spaces in-between. For example, the bitmap for the Lisa's system font looks like the following:

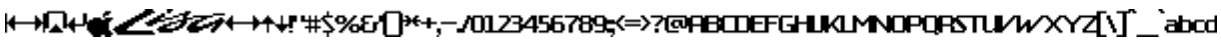

Figure 9 - Lisa system font bitmap

After the bitmap is a table of each character's X coordinate within the bitmap, followed by the offsets and widths table.

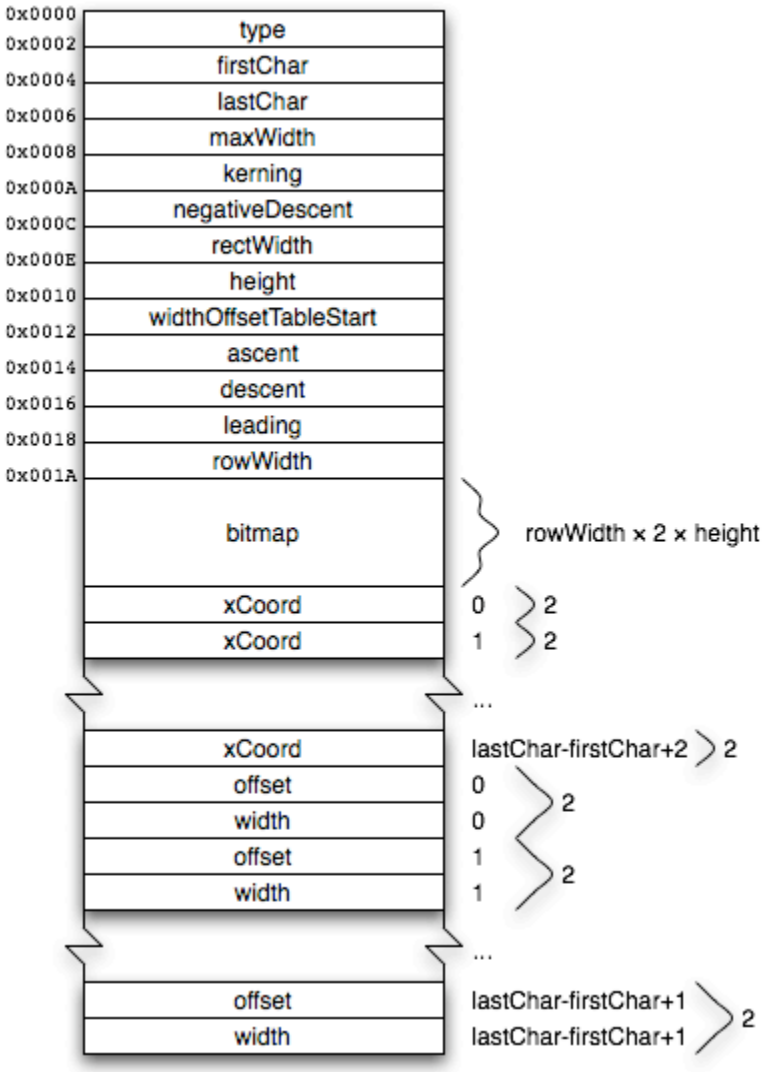

Figure 10 - Uncompressed bitmap font format

The offset to the offsets and widths table is expressed as the number of words after location `0x0010`. Always use the offset to find the table; do not rely on it immediately following the X coordinate table. The bitmap row width is expressed as the number of words in each row of the bitmap. NegativeDescent is simply the negation of the descent. Leading is the amount of vertical space between the bottom of one line of text and the top of the next line. *Inside Macintosh* describes kerning and rectWidth better than I ever could.

Each X coordinate in the table immediately following the bitmap is a word. There are (lastChar-firstChar+3) entries: one for each character in the range of defined characters, one for the catchall character, and one for the end of the bitmap. The end of the glyph for a character is determined to be the start of the glyph for the next character. If a character has no glyph, or is undefined, its X coordinate is the same as the next one.

There are (lastChar-firstChar+2) entries in the offsets and widths table, because the last entry is for the catchall character. The width is the number of columns of pixels the character occupies. The offset is the number of columns of pixels to the left of the character that are completely white. If a character in the middle of the range of defined characters is undefined, its width and offset are both -1 or `0xFF`.

Figure 11 demonstrates the meanings of these fields and tables graphically.

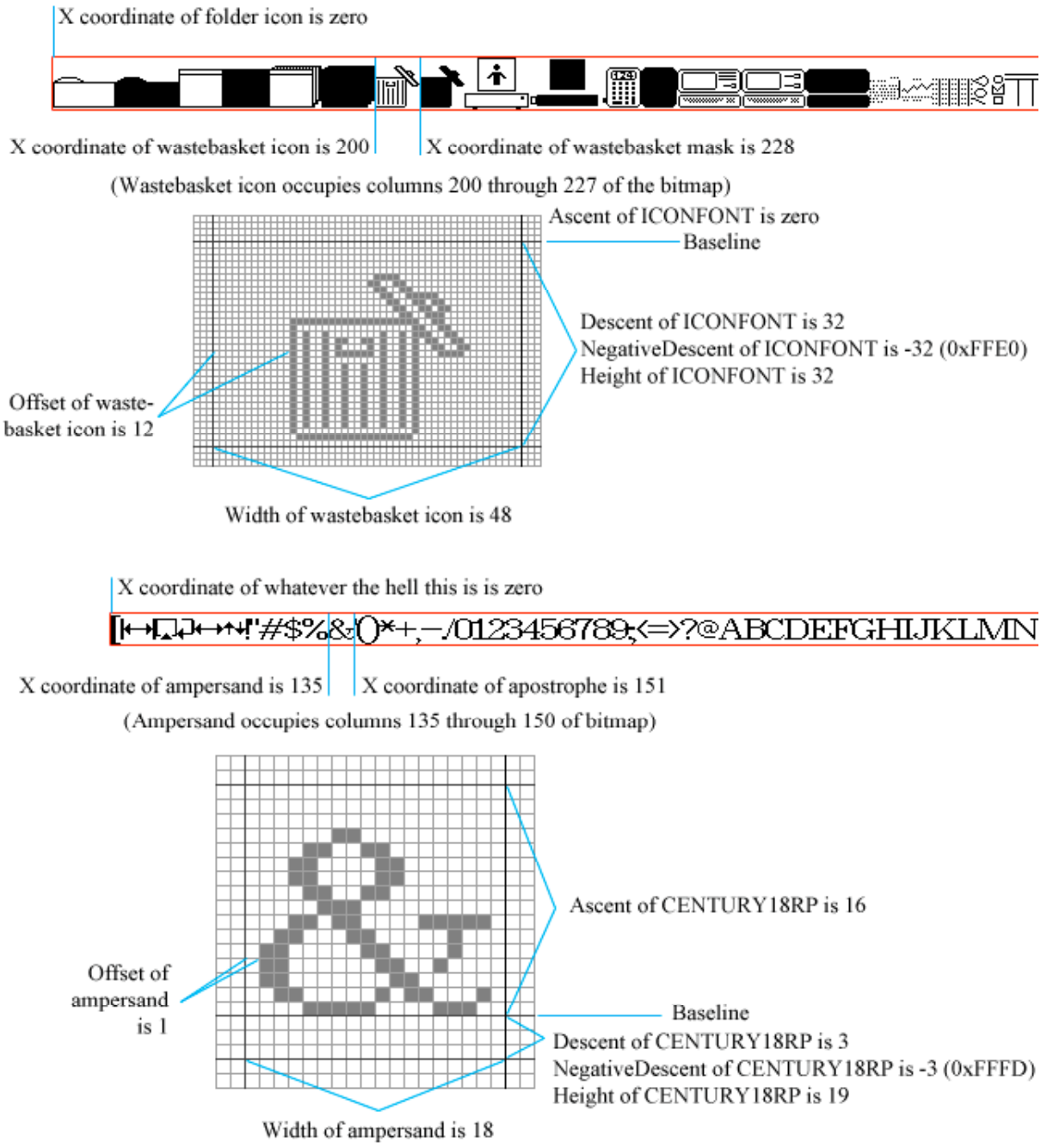

Figure 11 - Demonstration of font metrics

Appendix: The Code

Be Warned: This code is not guaranteed to be efficient. It is not guaranteed to have meaningful variable names. It will most definitely crash if given the wrong number of parameters, the name of a nonexistent file, or a file in a format it doesn't expect. It was written quickly during the reverse-engineering process and was intended only for that purpose. It is provided to help developers understand the data formats described in this document. If you're going to be reading Lisa fonts in an application you'll be using more than once, please write better code.

heur.c

The following is a C program to parse the `font.heur` file and present it in a human-readable form.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char * argv[])
{
	FILE * fp = fopen(argv[1], "rb");
	fseek(fp, 0, SEEK_END);
	int s = ftell(fp);
	fseek(fp, 0, SEEK_SET);
	unsigned char * stuff = (unsigned char *)malloc(s);
	fread(stuff, 1, s, fp);
	fclose(fp);

	int p = 0;
	int cnt = stuff[p++];
	printf("Count = %d\n", cnt);
	while (cnt) {
		printf("\n");
		int a0 = stuff[p++];
		int a1 = stuff[p++];
		int a2 = stuff[p++];
		int a3 = stuff[p++];
		int a4 = stuff[p++];
		int a5 = stuff[p++];
		int a6 = stuff[p++];
		int a7 = stuff[p++];
		int a8 = stuff[p++];
		int a9 = stuff[p++];
		printf("\tID Number = %d\n", a0);
		printf("\t--------------------------------\n");
		printf("\t|$%02X|$%02X|$%02X|$%02X|$%02X|$%02X|$%02X|$%02X|\n",
			a1, a2, a3, a4, a5, a6, a7, a8);
		printf("\t|%3d|%3d|%3d|%3d|%3d|%3d|%3d|%3d|\n",
			a1, a2, a3, a4, a5, a6, a7, a8);
		printf("\t--------------------------------\n");
		printf("\tCount = %d\n", a9);
		while (a9) {
			int b0 = stuff[p++];
			int b1 = stuff[p++];
			int b2 = stuff[p++];
			int b3 = stuff[p++];
			int b4 = stuff[p++];
			int b5 = stuff[p++];
			int b6 = stuff[p++];
			int b7 = stuff[p++];
```

```
                        printf("\t\t--------------------------------\n");
                        printf("\t\t|$%02X|$%02X|$%02X|$%02X|$%02X|$%02X|$%02X|$%02X|\n",
                               b0, b1, b2, b3, b4, b5, b6, b7);
                        printf("\t\t|%3d|%3d|%3d|%3d|%3d|%3d|%3d|%3d|\n",
                               b0, b1, b2, b3, b4, b5, b6, b7);
                        printf("\t\t--------------------------------\n");
                        int sl = stuff[p++];
                        unsigned char * ss = (unsigned char *)malloc(sl+1);
                        strncpy(ss, &stuff[p], sl);
                        ss[sl] = 0;
                        p += sl;
                        printf("\t\tName = %s\n", ss);
                        a9--;
                }
                cnt--;
        }

        return 0;
}
```

libf.c

This program does two things: first, it parses the header of the font.lib file and presents it in a human-readable form; second, it splits the file into its components and saves them to their own files, giving the font resources names like `font.lib.SYSTEXTS.F.f` and the font metrics names like `font.lib.SYSTEXTS.F.m`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char * argv[])
{
        FILE * fp = fopen(argv[1], "rb");
        fseek(fp, 0, SEEK_END);
        int s = ftell(fp);
        fseek(fp, 0, SEEK_SET);
        unsigned char * stuff = (unsigned char *)malloc(s);
        fread(stuff, 1, s, fp);
        fclose(fp);

        int p = 0;
        int a0 = stuff[p++];
        int a1 = stuff[p++];
        int len = (a0<<8|a1)<<1;
        printf("Length of Header = %04X\n", len);
        int a2 = stuff[p++];
        int a3 = stuff[p++];
        int cnt = a2<<8|a3;
        printf("Count = %d\n", cnt);
        while (cnt) {
                printf("\n");
                int sl = stuff[p++];
                unsigned char * ss = (unsigned char *)malloc(sl+1);
                strncpy(ss, &stuff[p], sl);
                ss[sl] = 0;
                p += sl;
                if (p & 1) p++;
                printf("\tName = %s\n", ss);
                int b0 = stuff[p++];
                int b1 = stuff[p++];
```

```
            int b2 = stuff[p++];
            int b3 = stuff[p++];
            int dstart = ((b0<<24|b1<<16|b2<<8|b3)<<1)+len+4;
            int b4 = stuff[p++];
            int b5 = stuff[p++];
            int b6 = stuff[p++];
            int b7 = stuff[p++];
            int dend = ((b4<<24|b5<<16|b6<<8|b7)<<1)+len+4;
            int b8 = stuff[p++];
            int b9 = stuff[p++];
            int bA = stuff[p++];
            int bB = stuff[p++];
            int mstart = ((b8<<24|b9<<16|bA<<8|bB)<<1)+len+4;
            int bC = stuff[p++];
            int bD = stuff[p++];
            int bE = stuff[p++];
            int bF = stuff[p++];
            int mend = ((bC<<24|bD<<16|bE<<8|bF)<<1)+len+4;
            printf("\tStart of Data = %08X\n", dstart);
            printf("\tEnd of Data = %08X\n", dend);
            printf("\tStart of Metrics = %08X\n", mstart);
            printf("\tEnd of Metrics = %08X\n", mend);
            unsigned char * df = (unsigned char *)malloc(sl+strlen(argv[1])+4);
            sprintf(df, "%s.%s.f", argv[1], ss);
            FILE * dfp = fopen(df, "wb");
            fwrite(&stuff[dstart], 1, dend-dstart, dfp);
            unsigned char * mf = (unsigned char *)malloc(sl+strlen(argv[1])+4);
            sprintf(mf, "%s.%s.m", argv[1], ss);
            FILE * mfp = fopen(mf, "wb");
            fwrite(&stuff[mstart], 1, mend-mstart, mfp);
            cnt--;
        }

        return 0;
}
```

m.c

The following is a C program to parse the `.m` files created by `libf.c` and present the font metrics information in a human-readable form. This can also be used on `.f` files, as long as they are uncompressed.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* You don't fully appreciate MacRoman until you try to translate it into 7-bit ASCII. */
const char* mr2asc[] = {
        "^@","^A","^B","^C","^D","^E","^F","^G","|<-","->|","^J","^K","(^)","<-'","(`(","Lsa",
        "^P","^Q","^R","^S","^T","^U","^V","^W","^X","^Y","^Z","^[","<-","->","/\\","\\/",
        " ", "!", "\"","#", "$", "%", "&", "\'","(", ")", "*", "+", ",", "-", ".", "/",
        "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", ":", ";", "<", "=", ">", "?",
        "@", "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", "{", "|", "}", "~", "[X]",
        "A:","Ao","C,","E'","N~","O:","U:","a'","a`","a^","a:","a~","ao","c,","e'","e`",
        "e^","e:","i'","i`","i^","i:","n~","o'","o`","o^","o:","o~","u'","u`","u^","u:",
        "+", "o", "c|","L-","SS","*","PP","ss","(R)","(C)","TM","'",":", "=/","AE","O/",
        "oo","+-","<=",">=","Y=","u", "d", "E", "P", "p", "S", "a", "o", "O", "ae","o/",
        "?", "!", "-.","v-","f", "~~","/\\","<<",">>",".","  ", "A`","A~","O~","OE","oe",
        "-", "--","``","''","`", "'", "/", "<>","y:","Y:","/", "C=","<", ">", "fi","fl",
        "++",".", ",", ",,","%o","A^","E^","A'","E:","E`","I'","I^","I:","I`","O'","O^",
```

```
        "(`(","O`","U'","U^","U`","i", "^", "~", "-", "u", ".", "o", ",", "\"","c", "v", "[?]"
};

int main(int argc, char * argv[])
{
        FILE * fp = fopen(argv[1], "rb");
        fseek(fp, 0, SEEK_END);
        int s = ftell(fp);
        fseek(fp, 0, SEEK_SET);
        unsigned char * stuff = (unsigned char *)malloc(s);
        fread(stuff, 1, s, fp);
        fclose(fp);

        int p = 0;
        int a1, a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int f3 = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int f4 = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int f5 = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int f6 = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int f7 = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int f8 = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int f9 = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int fA = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int fB = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int fC = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int fD = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int fE = a1<<8|a2;
        a1 = stuff[p++]; a2 = stuff[p++]; int fF = a1<<8|a2;
        printf("     Flags = %04X\n", f3, f3);
        printf("First Char = %02X\n", f4);
        printf(" Last Char = %02X\n", f5);
        printf(" Max Width = %d\n", f6);
        printf("   Kerning = %d\n", f7);
        printf("  NDescent = %d\n", (f8<0x8000)?(f8):(f8-0x10000));
        printf(" RectWidth = %d\n", f9);
        printf("    Height = %d\n", fA);
        printf("OfstToWTbl = %d\n", (fB<<1)+16);
        printf("    Ascent = %d\n", fC);
        printf("   Descent = %d\n", fD);
        printf("   Leading = %d\n", fE);
        printf("  RowWidth = %04X\n", fF, fF);
        p = (fB<<1)+16;
        while (f4 <= (f5+1)) {
                int b1 = stuff[p++];
                int b2 = stuff[p++];
                printf("Character %02X %-3s: Offset = %d, Width = %d\n",
                        f4, mr2asc[f4], b1, b2);
                f4++;
        }

        return 0;
}
```

f.c

The following program will decompress a compressed `.f` file. Unlike the previous programs, it takes two command line arguments: the name of the input file, and the name of the output file. These can be the same if you want to overwrite the compressed version.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
int main(int argc, char * argv[])
{
	FILE * fp = fopen(argv[1], "rb");
	fseek(fp, 0, SEEK_END);
	int s = ftell(fp);
	fseek(fp, 0, SEEK_SET);
	unsigned char * stuff = (unsigned char *)malloc(s);
	fread(stuff, 1, s, fp);
	fclose(fp);

	int p = 0;
	int a0 = stuff[p++];
	int a1 = stuff[p++];
	int a2 = stuff[p++];
	int a3 = stuff[p++];
	int a4 = stuff[p++];
	int a5 = stuff[p++];
	int a6 = stuff[p++];
	int a7 = stuff[p++];
	int a8 = stuff[p++];
	int a9 = stuff[p++];
	int   compLen = a2<<24 | a3<<16 | a4<<8 | a5;
	int decompLen = a6<<24 | a7<<16 | a8<<8 | a9;
	int decompLen8 = (decompLen & 7)?((decompLen|7)+1):decompLen;

	unsigned char * junk = (unsigned char *)malloc(2+decompLen8);
	junk[0] = a0;
	junk[1] = a1^0x80;
	memcpy(junk+2, stuff+10, compLen);
	int cp = 2+compLen;
	int dp = 2+decompLen8;
	while (cp > 2 && dp > 2) {
		int rc = (junk[--cp] | 0x100);
		while (rc >> 1) {
			if (rc & 1) junk[--dp] = 0;
			else        junk[--dp] = junk[--cp];
			rc >>= 1;
		}
	}
	int rw = (junk[0x18]<<9 | junk[0x19]<<1);
	int h  = (junk[0x0E]<<8 | junk[0x0F]);
	cp = 0x1A;
	dp = 0x1A+rw;
	int th = h;
	while (--th) {
		int trw = rw;
		while (trw--) junk[dp++] ^= junk[cp++];
	}

	FILE * dfp = fopen(argv[2], "wb");
	fwrite(junk, 1, decompLen+2, dfp);
	fclose(dfp);

	return 0;
}
```