

Devices_Manual

COLLABORATORS

	<i>TITLE :</i> Devices_Manual		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 18, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Devices_Manual	1
1.1	FORM Specs from the Original EA Document / FTXT IFF Formatted Text	1
1.2	FTXT IFF Formatted Text / Introduction	1
1.3	FTXT IFF Formatted Text / Reference	2
1.4	FTXT IFF Formatted / Standard Data and Property Chunks	2
1.5	Standard Data and Property Chunks / Character Set	3
1.6	Standard Data and Property Chunks / Control Sequences	3
1.7	Standard Data and Property Chunks / Data Chunk CHRS	4
1.8	Standard Data and Property Chunks / Property Chunk FONS	4
1.9	Standard Data and Property Chunks / Future Properties	5
1.10	Standard Data and Property Chunks / Positioning Units	5
1.11	FTXT IFF Formatted / FTXT Stripper	5
1.12	FTXT IFF Formatted / Appendix A: Character Code Table	8
1.13	FTXT IFF Formatted / Appendix B. FTXT Example	9
1.14	FTXT IFF Formatted / Appendix C. ISO/ANSI Control Sequences	9

Chapter 1

Devices_Manual

1.1 FORM Specs from the Original EA Document / FTXT IFF Formatted Text

Date: November 15, 1985 (Updated Oct, 1988 Commodore-Amiga, Inc.)
From: Steve Shaw and Jerry Morrison, Electronic Arts
and Bob "Kodiak" Burns, Commodore-Amiga
Status: Adopted

Introduction
Standard Data and Property Chunks
FTXT Stripper
Appendix A: Character Code Table
Appendix B: FTXT Example
Appendix C: ISO/ANSI Control Sequences

1.2 FTXT IFF Formatted Text / Introduction

This memo is the IFF supplement for FORM FTXT. An FTXT is an IFF "data section" or "FORM type" - which can be an IFF file or a part of one - containing a stream of text plus optional formatting information. EA IFF 85 is Electronic Arts' standard for interchange format files. (See the IFF reference.)

An FTXT is an archival and interchange representation designed for three uses. The simplest use is for a "console device" or "glass teletype" (the minimal 2-D text layout means): a stream of "graphic" ("printable") characters plus positioning characters "space" ("SP") and line terminator ("LF"). This is not intended for cursor movements on a screen although it does not conflict with standard cursor-moving characters. The second use is text that has explicit formatting information (or "looks") such as font family and size, typeface, etc. The third use is as the lowest layer of a structured document that also has "inherited" styles to implicitly control character looks. For that use, FORMs FTXT would be embedded within a future document FORM type. The beauty of FTXT is that these three uses are interchangeable, that is, a program written for one purpose can read and write the others' files. So a word processor does not have to write a separate plain text file to communicate with other programs.

Text is stored in one or more "CHRS" chunks inside an FTEXT. Each CHRS contains a stream of 8-bit text compatible with ISO and ANSI data interchange standards. FTEXT uses just the central character set from the ISO/ANSI standards. (These two standards are henceforth called "ISO/ANSI" as in "see the ISO/ANSI reference".)

Since it's possible to extract just the text portions from future document FORM types, programs can exchange data without having to save both plain text and formatted text representations.

Character looks are stored as embedded control sequences within CHRS chunks. This document specifies which class of control sequences to use: the CSI group. This document does not yet specify their meanings, e.g., which one means "turn on italic face". Consult ISO/ANSI.

Section 2 defines the chunk types character stream "CHRS" and font specifier "FONS". These are the "standard" chunks. Specialized chunks for private or future needs can be added later. Section 3 outlines an FTEXT reader program that strips a document down to plain unformatted text. Appendix A is a code table for the 8-bit ISO/ANSI character set used here. Appendix B is an example FTEXT shown as a box diagram. Appendix C is a racetrack diagram of the syntax of ISO/ANSI control sequences.

Reference

1.3 FTEXT IFF Formatted Text / Reference

Amiga® is a registered trademark of Commodore-Amiga, Inc.
Electronic Arts(tm) is a trademark of Electronic Arts.

IFF: "EA IFF 85" Standard for Interchange Format Files describes the underlying conventions for all IFF files.

ISO/ANSI: ISO/DIS 6429.2 and ANSI X3.64-1979. International Organization for Standardization (ISO) and American National Standards Institute (ANSI) data-interchange standards. The relevant parts of these two standards documents are identical. ISO standard 2022 is also relevant.

1.4 FTEXT IFF Formatted / Standard Data and Property Chunks

The main contents of a FORM FTEXT is in its character stream "CHRS" chunks. Formatting property chunks may also appear. The only formatting property yet defined is "FONS", a font specifier. A FORM FTEXT with no CHRS represents an empty text stream. A FORM FTEXT may contain nested IFF FORMs, LISTs, or CAT s, although a "stripping" reader (see section 3) will ignore them.

Character Sets
Control Sequences
Data Chunk CHRS
Property Chunk FONS
Future Properties

Positioning Units

1.5 Standard Data and Property Chunks / Character Set

FORM FTXT uses the core of the 8-bit character set defined by the ISO/ANSI standards cited at the start of this document. (See Appendix A for a character code table.) This character set is divided into two "graphic" groups plus two "control" groups. Eight of the control characters begin ISO/ANSI standard control sequences. (See "Control Sequences", below.) Most control sequences and control characters are reserved for future use and for compatibility with ISO/ANSI. Current reader programs should skip them.

- * C0 is the group of control characters in the range NUL (hex 0) through hex 1F. Of these, only LF (hex 0A) and ESC (hex 1B) are significant. ESC begins a control sequence. LF is the line terminator, meaning "go to the first horizontal position of the next line". All other C0 characters are not used. In particular, CR (hex 0D) is not recognized as a line terminator.
- * G0 is the group of graphic characters in the range hex 20 through hex 7F. SP (hex 20) is the space character. DEL (hex 7F) is the delete character which is not used. The rest are the standard ASCII printable characters "!" (hex 21) through "~" (hex 7E).
- * C1 is the group of extended control characters in the range hex 80 through hex 9F. Some of these begin control sequences. The control sequence starting with CSI (hex 9B) is used for FTXT formatting. All other control sequences and C1 control characters are unused.
- * G1 is the group of extended graphic characters in the range NBSP (hex A0) through "ÿ" (hex FF). It is one of the alternate graphic groups proposed for ISO/ANSI standardization.

1.6 Standard Data and Property Chunks / Control Sequences

Eight of the control characters begin ISO/ANSI standard "control sequences" (or "escape sequences"). These sequences are described below and diagramed in Appendix C.

```
G0    ::= (SP through DEL)
G1    ::= (NBSP through "ÿ")
```

```
ESC-Seq ::= ESC (SP through "/" ) * ("0" through "~")
ShiftToG2 ::= SS2 G0
ShiftToG3 ::= SS3 G0
CSI-Seq  ::= CSI (SP through "?") * ("@" through "~")
DCS-Seq  ::= (DCS | OSC | PM | APC) (SP through "~" | G1) * ST
```

"ESC-Seq" is the control sequence ESC (hex 1B), followed by zero or more characters in the range SP through "/" (hex 20 through hex 2F), followed

by a character in the range "0" through "~" (hex 30 through hex 7E). These sequences are reserved for future use and should be skipped by current FTXT reader programs.

SS2 (hex 8E) and SS3 (hex 8F) shift the single following G0 character into yet-to-be-defined graphic sets G2 and G3, respectively. These sequences should not be used until the character sets G2 and G3 are standardized. A reader may simply skip the SS2 or SS3 (taking the following character as a corresponding G0 character) or replace the two-character sequence with a character like "?" to mean "absent".

FTXT uses "CSI-Seq" control sequences to store character formatting (font selection by number, type face, and text size) and perhaps layout information (position and rotation). "CSI-Seq" control sequences start with CSI (the "control sequence introducer", hex 9B). Syntactically, the sequence includes zero or more characters in the range SP through "?" (hex 20 through hex 3F) and a concluding character in the range "@" through "~" (hex 40 through hex 7E). These sequences may be skipped by a minimal FTXT reader, i.e., one that ignores formatting information.

Note: A future FTXT standardization document will explain the uses of CSI-Seq sequences for setting character face (light weight vs. medium vs. bold, italic vs. upright, height, pitch, position, and rotation). For now, consult the ISO/ANSI references.

"DCS-Seq" is the control sequences starting with DCS (hex 90), OSC (hex 9D), PM (hex 9E), or APC (hex 9F), followed by zero or more characters each of which is in the range SP through "~" (hex 20 through hex 7E) or else a G1 character, and terminated by an ST (hex 9C). These sequences are reserved for future use and should be skipped by current FTXT reader programs.

1.7 Standard Data and Property Chunks / Data Chunk CHRS

A CHRS chunk contains a sequence of 8-bit characters abiding by the ISO/ANSI standards cited at the start of this document. This includes the character set and control sequences as described above and summarized in Appendix A and Appendix C.

A FORM FTXT may contain any number of CHRS chunks. Taken together, they represent a single stream of textual information. That is, the contents of CHRS chunks are effectively concatenated except that (1) each control sequence must be completely within a single CHRS chunk, and (2) any formatting property chunks appearing between two CHRS chunks affects the formatting of the latter chunk's text. Any formatting settings set by control sequences inside a CHRS carry over to the next CHRS in the same FORM FTXT. All formatting properties stop at the end of the FORM since IFF specifies that adjacent FORMs are independent of each other (although not independent of any properties inherited from an enclosing LIST or FORM).

1.8 Standard Data and Property Chunks / Property Chunk FONS

The optional property "FONS" holds a FontSpecifier as defined in the C declaration below. It assigns a font to a numbered "font register" so it can be referenced by number within subsequent CHRS chunks. (This function is not provided within the ISO and ANSI standards.) The font specifier gives both a name and a description for the font so the recipient program can do font substitution.

By default, CHRS text uses font 1 until it selects another font. A minimal text reader always uses font 1. If font 1 hasn't been specified, the reader may use the local system font as font 1.

```
typedef struct {
    UBYTE id;          /* 0 through 9 is a font id number referenced by an SGR
                        control sequence selective parameter of 10 through 19.
                        Other values are reserved for future standardization. */
    UBYTE pad1;        /* reserved for future use; store 0 here */
    UBYTE proportional; /* proportional font-- 0=unknown, 1=no, 2=yes */
    UBYTE serif;        /* serif font-- 0 = unknown, 1 = no, 2 = yes */
    char name[];        /* A NUL-terminated string naming the preferred font. */
} FontSpecifier;
```

Fields are filed in the order shown. The UBYTE fields are byte-packed (2 per 16-bit word). The field pad1 is reserved for future standardization. Programs should store 0 there for now.

The field proportional indicates if the desired font is proportional width as opposed to fixed width. The field serif indicates if the desired font is serif as opposed to sans serif. [Issue: Discuss font substitution!]

1.9 Standard Data and Property Chunks / Future Properties

New optional property chunks may be defined in the future to store additional formatting information. They will be used to represent formatting not encoded in standard ISO/ANSI control sequences and for "inherited" formatting in structured documents. Text orientation might be one example.

1.10 Standard Data and Property Chunks / Positioning Units

Unless otherwise specified, position and size units used in FTXT formatting properties and control sequences are in decipoints (720 decipoints/inch). This is ANSI/ISO Positioning Unit Mode (PUM) 2. While a metric standard might be nice, decipoints allow the existing U.S.A. typographic units to be encoded easily, e.g., "12 points" is "120 decipoints".

1.11 FTXT IFF Formatted / FTXT Stripper

An FTEXT reader program can read the text and ignore all formatting and structural information in a document FORM that uses FORMs FTEXT for the leaf nodes. This amounts to stripping a document down to a stream of plain text. It would do this by skipping over all chunks except FTEXT.CHRS (CHRS chunks found inside a FORM FTEXT) and within the FTEXT.CHRS chunks skipping all control characters and control sequences. (Appendix C diagrams this text scanner.) It may also read FTEXT.FONS chunks to find a description for font 1.

Here's a Pascal-ish program for an FTEXT stripper. Given a FORM (a document of some kind), it scans for all FTEXT.CHRS chunks. This would likely be applied to the first FORM in an IFF file.

```

PROCEDURE ReadFORM4CHRS();      {Read an IFF FORM for FTEXT.CHRS chunks.}
BEGIN
  IF the FORM's subtype = "FTEXT"
    THEN ReadFTEXT4CHRS()
  ELSE WHILE something left to read in the FORM
    DO BEGIN
      read the next chunk header;
      CASE the chunk's ID OF
        "LIST", "CAT ": ReadCAT4CHRS();
        "FORM": ReadFORM4CHRS();
        OTHERWISE skip the chunk's body;
      END
    END
  END;

  {Read a LIST or CAT for all FTEXT.CHRS chunks.}
  PROCEDURE ReadCAT4CHRS();
  BEGIN
    WHILE something left to read in the LIST or CAT
    DO BEGIN
      read the next chunk header;
      CASE the chunk's ID OF
        "LIST", "CAT ": ReadCAT4CHRS();
        "FORM": ReadFORM4CHRS();
        "PROP": IF we're reading a LIST AND the PROP's subtype = "FTEXT"
          THEN read the PROP for "FONS" chunks;
        OTHERWISE error--malformed IFF file;
      END
    END
  END;

  PROCEDURE ReadFTEXT4CHRS();    {Read a FORM FTEXT for CHRS chunks.}
  BEGIN
    WHILE something left to read in the FORM FTEXT
    DO BEGIN
      read the next chunk header;
      CASE the chunk's ID OF
        "CHRS": ReadCHRS();
        "FONS": BEGIN
          read the chunk's contents into a FontSpecifier variable;
          IF the font specifier's id = 1
            THEN use this font;
          END;
      END;
    END;
  END;

```

```

        OTHERWISE skip the chunk's body;
    END
END
END;

{Read an FTXT.CHRS. Skip all control sequences and unused control chars.}
PROCEDURE ReadCHRS();
BEGIN
    WHILE something left to read in the CHRS chunk
    DO
        CASE read the next character OF
            LF: start a new output line;
            ESC: SkipControl([' '..'/'], ['0'..'~']);
            IN [' '..'~'], IN [NBSP..'ÿ']: output the character;
            SS2, SS3: ; {Just handle the following G0 character directly,
                        ignoring the shift to G2 or G3.}
            CSI: SkipControl([' '..'?'], ['@'..'~']);
            DCS, OSC, PM, APC: SkipControl([' '..'~'] + [NBSP..'ÿ'], [ST]);
        END
    END
END;

{Skip a control sequence of the format (rSet)* (tSet), i.e., any number of
characters in the set rSet followed by a character in the set tSet.}
PROCEDURE SkipControl(rSet, tSet);
VAR c: CHAR;
BEGIN
    REPEAT c := read the next character
    UNTIL c NOT IN rSet;

    IF c NOT IN tSet
    THEN put character c back into the input stream;
    END
END

```

The following program is an optimized version of the above routines ReadFORM4CHRS and ReadCAT4CHRS for the case where you're ignoring fonts as well as formatting. It takes advantage of certain facts of the IFF format to read a document FORM and its nested FORMs, LISTs, and CAT s without a stack. In other words, it's a hack that ignores all fonts and faces to cheaply get to the plain text of the document.

```

{Cheap scan of an IFF FORM for FTXT.CHRS chunks.}
PROCEDURE ScanFORM4CHRS();
BEGIN
    IF the document FORM's subtype = "FTXT"
    THEN ReadFTXT4CHRS()
    ELSE WHILE something left to read in the FORM
    DO BEGIN
        read the next chunk header;
        IF it's a group chunk (LIST, FORM, PROP, or CAT)
        THEN read its subtype ID;
        CASE the chunk's ID OF
            "LIST", "CAT ";; {NOTE: See explanation below.*}

```

```

"FORM": IF this FORM's subtype = "FTXT"
    THEN ReadFTXT4CHRS()
    ELSE; {NOTE: See explanation below.*}
    OTHERWISE skip the chunk's body;
END
END
END;

```

*Note: This implementation is subtle. After reading a group header other than FORM FTXT it just continues reading. This amounts to reading all the chunks inside that group as if they weren't nested in a group.

1.12 FTXT IFF Formatted / Appendix A: Character Code Table

This table corresponds to the ISO/DIS 6429.2 and ANSI X3.64-1979 8-bit character set standards. Only the core character set of those standards is used in FTXT.

Two G1 characters aren't defined in the standards and are shown as dark gray entries in this table. Light gray shading denotes control characters. (DEL is a control character although it belongs to the graphic group G0.)

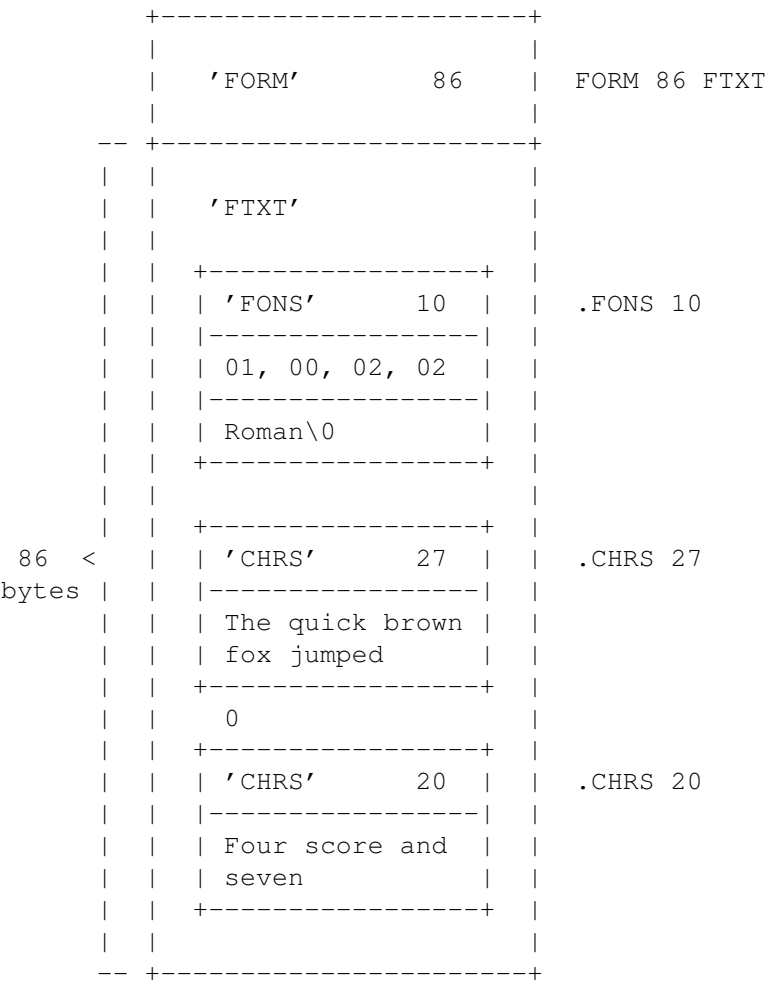
ISO/DIS 6429.2 and ANSI X3.64-1979 Character Code Table

MSN (most significant nybble)																					
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F				
+		-----+							+		-----										
LSN	0	NUL		SP	0	@	P	`	p		DCS	NBSP	\textdegree{}		À		Đ	à	←		
	1	ø												\ensuremath{\backslash pm}		Á		Ñ	←		
		á	ñ		!	1	A	Q	a	q											
	2				"	2	B	R	b	r				ç	\$^2\$	Â	Ò	â	ò		
	3				#	3	C	S	c	s				£	\$^3\$	Ã	Ó	ã	ó		
	4				\$	4	D	T	d	t				¤		Ä	Ô	ä	ô		
	5				%	5	E	U	e	u				\$\yen\$		\$\\mathrm{\mu}\$		\$	Å	←	
		Ö	ä	ö																	
	6				&	6	F	V	f	v					¶	Æ	Ö	æ	ö		
	7				'	7	G	W	g	w				\$	·	Ç	\$\\times\$		ç	\$\\	←
		div\$																			
	8				(8	H	X	h	x				uml	,	È	Ø	è	ø		
	9)	9	I	Y	i	y				(c)	\$^1\$	É	Ù	é	ù		
A	LF				*	:	J	Z	j	z					°	Ê	Û	ê	û		
B	ESC				+	;	K	[k	{		CSI		«	»	Ë	Ü	ë	ü		
C					,	<	L	\	l			ST		\ensuremath{\backslash not}				¼	İ	←	
		Ü	ì	ü																	
D	CR				-	=	M]	m	}		OSC	SHY	½	Í	Ý	í	ý			
E					.	>	N	^	n	~	SS2	PM	®	¾	Î	Þ	î	þ			
F					/	?	O	_	o	DEL	SS3	APC		¿	Ï	ß	ï	ÿ			
_ _ /		_ _ _ _ /				_ _ _ _ /				_ _ /		_ _ _ _ _ _ /									
V		V				V				V		V									
Control Group C0		Graphic Group G0				Control Group C1				Graphic Group G1											

NBSP is non-breaking space
SHY is soft hyphen

1.13 FTXT IFF Formatted / Appendix B. FTXT Example

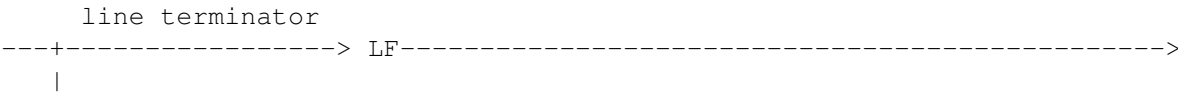
Here's a box diagram for a simple example: "The quick brown fox jumped.
Four score and seven", written in a proportional serif font named "Roman".



The "0" after the first CHRS chunk is a pad byte.

1.14 FTXT IFF Formatted / Appendix C. ISO/ANSI Control Sequences

This is a racetrack diagram of the ISO/ANSI characters and control sequences as used in FTXT CHRS chunks.



```

|\  ESC-Seq
| +-----> ESC-----+-----> 0 through ~ --->
|           |           |
|           +<----SP through \  <----+
|
|\  printable
| +-----+-----> SP through ~ ----->
|           |           /
|           +-----> G1----->+
|
|\  ShiftToG2
| +-----> G0-----> (produces a G2 character)
|
|\  ShiftToG3
| +-----> G0-----> (produces a G3 character)
|
|\  CSI-Seq
| +-----> CSI--+-----+-----> @ through ~ --->
|           |           |
|           +<---- SP through ?  <----+
|
|\  DCS-Seq
| +-----> DCS, OSC,--+-----+-----> ST ----->
|           PM, APC  |           | \
|           +<---- SP through ~  <----+   +-----> G1 ----->
|
|\  discard
| +-----> any other character----->

```

Of the various control sequences, only CSI-Seq is used for FTXT character formatting information. The others are reserved for future use and for compatibility with ISO/ANSI standards. Certain character sequences are syntactically malformed, e.g., CSI followed by a C0, C1, or G1 character. Writer programs should not generate reserved or malformed sequences and reader programs should skip them.

Consult the ISO/ANSI standards for the meaning of the CSI-Seq control sequences.

The two character set shifts SS2 and SS3 may be used when the graphic character groups G2 and G3 become standardized.