Fonts in XFree86

Juliusz Chroboczek, Juliusz.Chroboczek@ens.fr

5 March 2000

XFree86 contains a number of improvements related to fonts, including

- inclusion of new fonts;
- internationalisation of the scalable font backends (Type 1, Speedo, and TrueType);
- support for TrueType fonts;
- support for CID-keyed fonts.

This document describes these improvements. It does not attempt to describe the standard support for fonts in X11; the reader is referred to the `X`(1), `Xserver`(1), and `mkfontdir`(1) manpages.

1. Background and terminology

1.1 Characters and glyphs

A *character* is an abstract unit of a writing system. Examples of characters include the Latin capital letter *A*, the Arabic letter *jim*, and the *dingbat black scissors*.

A *glyph* is a shape that may represent one or many characters when displayed by a window system or printed by a printer.

While glyphs roughly correspond to characters in most cases, this correspondence is not, in general, one to one. For example, a font may have many variant forms of the capital letter *A*; a single *fi* ligature may correspond to the letters *f* and *i*.

A *coded character set* is a set of characters together with a mapping from integer codes -- known as *codepoints* -- to characters. Examples of coded character sets include US-ASCII, ISO 8859-1, KOI8-R, and JIS X 0208(1990).

A coded character set need not use 8-bit integers to index characters. Many early mainframes used 6-bit character sets, while 16-bit (or more) character sets are necessary for ideographic writing systems.

1.2 Font files, fonts, and XLFD

Traditionally, typographers speak about *typefaces* and *founts* (we use the traditional British spelling to distinguish founts from digital fonts). A typeface is a particular style or design, such as Times Italic, while a fount is a molten-lead incarnation of a given typeface at a given size.

Digital fonts come in *font files*. A font file contains all the information necessary for generating glyphs of a given typeface, and applications using font files may access glyph information in arbitrary order.

Digital fonts may consist of bitmap data, in which case they are said to be *bitmap fonts*. They may also consist of a mathematical description of glyph shapes, in which case they are said to be *scalable fonts*. Common formats for scalable font files are *Type 1* (sometimes incorrectly called *ATM fonts* or *PostScript fonts*), *Speedo* and *TrueType*.

The glyph data in a digital font needs to be indexed somehow. How this is done depends on the font file format. In the case of Type 1 fonts, glyphs are identified by *glyph names*. In the case of TrueType fonts, glyphs are indexed by integers corresponding to one of a number of indexing schemes (usually Unicode --- see below).

The X11 system uses the data in font file to generate *font instances*, which are collections of glyphs at a given size indexed according to a given encoding. X11 font instances are specified using a notation known as the *X Logical Font Description* (XLFD). An XLFD starts with a dash '-', and consists of fourteen fields separated by dashes, for example

```
-adobe-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1
```

Or particular interest are the last two fields '`iso8859-1`', which specify the font instance's encoding.

1.3 Unicode

Unicode (`<URL:http://www.unicode.org>`) is a coded character set with the goal of uniquely identifying all characters for all scripts, current and historical. While Unicode was explicitly not designed as a glyph encoding scheme, it is often possible to use it as such.

Unicode is an *open* character set, in that codepoint assignments may be added to Unicode at any time (once specified, though, an assignment can never be changed). For this reason, a Unicode font will be *sparse*, and only define glyphs for a subset of the character registry of Unicode.

The Unicode standard is defined in parallel with ISO 10646. Assignments in the two standards are always equivalent, and this document uses the terms ''Unicode'' and ''ISO 10646'' interchangeably.

When used in X11, Unicode-encoded fonts should have the last two fields of their XLFD set to '`iso10646-1`'.

2. New fonts

2.1 Bitmap fonts

XFree86 includes two new Unicode-encoded fonts with a large collection of non-ideographic glyphs. While it is possible to use these fonts as main fonts, applications may also use them as fallbacks when a given glyph is not available in the current font.

2.1.1 The Unicode 'fixed' font

The font file

```
/usr/X11/lib/X11/fonts/misc/6x13.pcf.gz
```

with XLFD

```
-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso10646-1
```

is a Unicode-encoded version of the standard '`fixed`' font with added support for the Latin, Greek, Cyrillic, Georgian, Armenian, IPA and other scripts plus numerous technical symbols. It contains over 2800 characters, covering all characters of ISO 8859 parts 1-5, 7-10, 13-15, as well as all European IBM and Microsoft code pages, KOI8, WGL4, and the repertoires of many other character sets. This font is compatible with the standard 8-bit `fixed` font and therefore also includes the DEC line-drawing glyphs in the range 0x00 to 0x1F, which are not part of Unicode or ISO 10646-1.

An ISO 8859-1 version of this font is still available in file

```
/usr/X11/lib/X11/fonts/misc/6x13-ISO8859-1.pcf.gz
```

with XLFD

```
-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
```

The standard aliases '`fixed`' and '`6x13`' still point at the ISO 8859-1 versions of the font.

2.1.2 The ClearlyU Unicode font

The ClearlyU font set of fonts provides a set of 12pt, 100dpi proportional fonts with many of the glyphs needed for Unicode text. Together, the fonts contain over 4000 glyphs.

The main ClearlyU font has XLFD name

```
-mutt-ClearlyU-medium-r-normal--17-120-100-100-p-101-iso10646-1
```

and resides in the font file

```
/usr/X11/lib/X11/fonts/misc/cu12.pcf.gz
```

Additional ClearlyU fonts include

```
-mutt-ClearlyU Alternate Glyphs-medium-r-normal--17-120-100-100-p-91-iso10646-1
-mutt-ClearlyU Arabic Extra-medium-r-normal--17-120-100-100-p-103-fontspecific-0
-mutt-ClearlyU Ligature-medium-r-normal--17-120-100-100-p-141-fontspecific-0
-mutt-ClearlyU PUA-medium-r-normal--17-120-100-100-p-111-iso10646-1
```

2.2 Scalable fonts

XFree86 includes the ''Lucidux'' family of Type 1 fonts. This family consists of the fonts ''Lucidux Serif'', with XLFD

```
-b&h-lucidux serif-medium-*-normal--*-*-*-*-p-*-*-*
```

''Lucidux Sans'', with XLFD

```
-b&h-lucidux sans-medium-*-normal--*-*-*-*-p-*-*-*
```

and ''Lucidux Mono'', with XLFD

```
-b&h-lucidux mono-medium-*-normal--*-*-*-*-m-*-*-*
```

Each of these fonts currently comes in Roman and oblique variants (bold variants will be included in a future release) and has 337 glyphs covering the basic ''ASCII'' glyph set, the Latin 1 glyph set, as well as the ''Extended Latin'' glyph set. In particular, these fonts include all the glyphs needed for ISO 8859 parts 1, 2, 3, 4, 9 and 15.

The Lucidux fonts are original designs by Charles Bigelow and Kris Holmes. Lucidux fonts include seriffed, sans-serif, and monospaced styles which share the same stem weight, x-height, capital height, ascent and descent. Lucidux fonts harmonise with Lucida (R) fonts of the same vertical proportions and weights. The character width metrics of Lucidux roman fonts match those of core fonts bundled with several window systems.

Each PFA file has a copy of the license terms in PS comment lines. The license terms are also included in the file `COPYRIGHT.BH` for convenience, and in the License document.

The design and font outlines were donated by Charles Bigelow and Kris Holmes from Bigelow and Holmes Inc., and the hinting was donated by Berthold Horn and Blenda Horn from Y&Y, Inc. For more information, please contact <design@bigelowandholmes.com> or <sales@yandy.com>, or consult Y&Y's web site <URL:http://www.yandy.com>.

3. Internationalisation of scalable font backends.

The scalable font backends (Type 1, Speedo, TrueType) can now automatically re-encode fonts to the encoding specified in the XLFD in '`fonts.dir`'. For example, a '`fonts.dir`' file can now contain entries for the Type 1 Courier font such as

```
cour.pfa -adobe-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1
cour.pfa -adobe-courier-medium-r-normal--0-0-0-0-m-0-iso8859-2
```

which will lead to the font being recoded to ISO 8859-1 and ISO 8859-2 respectively.

3.1 The 'fontenc' layer

Three of the scalable backends (Type 1, Speedo, and the 'freetype' TrueType backend) use a common 'fontenc' layer for font re-encoding. This allows those backends to share their encoding data, and allows simple configuration of new locales independently of font type.

Please note: the X-TrueType (X-TT) backend does not use the 'fontenc' layer, but instead uses its own method for font reencoding. Readers only interested in X-TT may want to skip to *Using Symbol Fonts* (section 3.5, page 8), as the intervening information does not apply to X-TT. X-TT itself is described in more detail in *X-TrueType* (section 4.1.2, page 10).

In the 'fontenc' layer, an encoding is defined by a name (such as '`iso8859-1`'), eventually a number of aliases (alternate names), and an ordered collection of mappings. A mapping defines the way the encoding can be mapped into one of the "target" encodings known to the 'fontenc' layer; currently, those consist of Unicode, Adobe glyph names, and arbitrary TrueType 'cmap's.

A number of encodings are hardwired into 'fontenc', and are therefore always available; the hardcoded encodings cannot easily be redefined. These include:

- '`iso10646-1`': Unicode;
- '`iso8859-1`': ISO Latin-1 (Western Europe);
- '`iso8859-2`': ISO Latin-2 (Eastern Europe);
- '`iso8859-3`': ISO Latin-3 (Southern Europe);
- '`iso8859-4`': ISO Latin-4 (Northern Europe);
- '`iso8859-5`': ISO Cyrillic;
- '`iso8859-6`': ISO Arabic;
- '`iso8859-7`': ISO Greek;
- '`iso8859-8`': ISO Hebrew;
- '`iso8859-9`': ISO Latin-5 (Turkish);
- '`iso8859-10`': ISO Latin-6 (Nordic);

- 'iso8859-15': ISO Latin-9, or Latin-0 (Revised Western-European);
- 'koi8-r': KOI8 Russian;
- 'koi8-u': KOI8 Ukrainian (see RFC 2319);
- 'koi8-ru': KOI8 Russian/Ukrainian
- 'koi8-uni': KOI8 "Unified" (Russian, Ukrainian, and Byelorussian);
- 'koi8-e': KOI8 'European', ISO-IR-111, or ECMA-Cyrillic;
- 'microsoft-symbol' and 'apple-roman': these are only likely to be useful with TrueType symbol fonts.

New encodings can be added by defining *encoding files*. When a font encoding is requested that the 'fontenc' layer doesn't know about, the backend checks the directory in which the font file resides (not the directory with 'fonts.dir'!) for a file named 'encodings.dir'. If found, this file is scanned for the unknown encoding, and the requested encoding definition file is read in. The mkfontdir(1) utility, when invoked with the '-e' option followed by the name of a directory containing encoding files, can be used to automatically build 'encodings.dir' files. See the mkfontdir(1) manpage for more details.

A number of predefined encoding files have been included with the distribution. Information on writing new encoding files can be found in *Format of encodings directory files* (section 3.3, page 5) and *Format of encodings files* (section 3.4, page 6).

3.2 Backend-specific notes about fontenc

3.2.1 Type 1

The Type 1 backend first searches for a mapping with a target of PostScript. If one is found, it is used. If none is found, the backend searches for a mapping with target Unicode, which is then composed with a built-in table mapping codes to glyph names. Note that this table only covers part of the Unicode code points that have been assigned names by Adobe.

If neither a PostScript or Unicode mapping is found, the backend defaults to ISO 8859-1.

Specifying an encoding value of 'adobe-fontspecific' disables the encoding mechanism. This is useful with symbol and wrongly encoded fonts (see below).

The Type 1 backend currently limits all encodings to 8-bit codes.

3.2.2 Speedo

The Speedo backend searches for a mapping with a target of Unicode, and uses it if found. If none is found, the backend defaults to ISO 8859-1.

The Speedo backend limits all encodings to 8-bit codes.

3.2.3 The 'freetype' TrueType backend

The TrueType backend scans the mappings in order. Mappings with a target of PostScript are ignored; mappings with a TrueType or Unicode target are checked against all the cmaps in the file. The first applicable mapping is used.

Authors of encoding files to be used with the TrueType backend should ensure that mappings are mentioned in decreasing order of preference.

3.3 Format of encodings directory files

In order to use a font in an encoding that the font backend does not know about, you need to have a 'encodings.dir' file in the same directory as the font file used. 'encodings.dir' has the same format as 'fonts.dir'. Its first line specifies the number of encodings, while every successive line has two columns, the name of the encoding, and the name of the encoding file; this can be relative to the current directory, or absolute. Every encoding name should agree with the encoding name defined in the encoding file. For example,

```
3

mulearabic-0 encodings/mulearabic-0.enc

mulearabic-1 encodings/mulearabic-1.enc

mulearabic-2 encodings/mulearabic-2.enc
```

Note that the name of an encoding must be specified in the encoding file's STARTENCODING or ALIAS line. It is not enough to create an '`encodings.dir`' entry.

If your platform supports it (it probably does), encoding files may be compressed or gzipped.

'`encoding.dir`' files are best maintained by the `mkfontdir`(1) utility. Please see the `mkfontdir`(1) manpage for more information.

3.4 Format of encoding files

The encoding files are "free form," *i.e.* any string of whitespace is equivalent to a single space. Keywords are parsed in a non-case-sensitive manner, meaning that '`size`', '`SIZE`', and '`SiZE`' all parse as the same keyword; on the other hand, case is significant in glyph names.

Numbers can be written in decimal, as in '`256`', in hexadecimal, as in '`0x100`', or in octal, as in '`0400`'.

Comments are introduced by a hash sign '`#`'. A '`#`' may appear at any point in a line, and all characters following the '`#`' are ignored, up to the end of the line.

The encoding file starts with the definition of the name of the encoding, and eventually its alternate names (aliases):

```
STARTENCODING mulearabic-0

ALIAS arabic-0

ALIAS something-else
```

The names of the encoding should be suitable for use in an XLFD font name, and therefore contain exactly one dash '`-`'.

The encoding file may then optionally declare the size of the encoding. For a linear encoding (such as Mule Arabic, or ISO 8859-1), the SIZE line specifies the maximum code plus one:

```
SIZE 0x2B
```

For a matrix encoding, it should specify two numbers. The first is the number of the last row plus one, the other, the highest column number plus one. For example, in the case of '`jisx0208.1990-0`' (JIS X 0208(1990), double-byte encoding, high bit clear), it should be

```
SIZE 0x75 0x80
```

Codes outside the region defined by the size line are supposed to be undefined. Encodings default to linear encoding with a size of 256 (0x100). This means that you must declare the size of all 16 bit encodings.

What follows is one or more mapping sections. A mapping section starts with a '`STARTMAPPING`' line stating the target of the mapping. The target may be one of:

- Unicode (ISO 10646):

```
STARTMAPPING unicode
```

- a given TrueType 'cmap':

```
STARTMAPPING cmap 3 1
```

- PostScript glyph names

```
STARTMAPPING postscript
```

Every line in a mapping section maps one from the encoding being defined to the target of the mapping. In mappings with a Unicode or TrueType mapping, codes are mapped to codes:

```
0x21 0x0660
0x22 0x0661
...
```

As an abbreviation, it is possible to map a contiguous range of codes in a single line. A line consisting of three integers

start end target

is an abbreviation for the range of lines

```
start     target
start+1   target+1
...
end       target+end-start
```

For example, the line

```
0x2121 0x215F 0x8140
```

is an abbreviation for

```
0x2121 0x8140
0x2122 0x8141
...
0x215F 0x817E
```

Codes not listed are assumed to map through the identity (*i.e.* to the same numerical value). In order to override this default mapping, you may specify a range of codes to be undefined by using an 'UNDEFINE' line:

```
UNDEFINE 0x00 0x2A
```

or, for a single code

```
UNDEFINE 0x1234
```

This works because later values override earlier one.

PostScript mappings are different. Every line in a PostScript mapping maps a code to a glyph name

```
0x41 A
0x42 B
...
```

and codes not explicitly listed are undefined.

A mapping section ends with an `ENDMAPPING` line

```
ENDMAPPING
```

After all the mappings have been defined, the file ends with an `ENDENCODING` line

```
ENDENCODING
```

Lines of the form

```
UNASSIGNED 0x00 0x1F
```

or

```
UNASSIGNED 0x1234
```

are ignored by the server, but may be used by supporting utilities.

In order to make future extensions to the format possible, lines starting with an unknown keyword are ignored, as are mapping sections with an unknown target.

3.5 Using symbol fonts

Type 1 symbol fonts should be installed using the '`adobe-fontspecific`' encoding.

In an ideal world, all TrueType symbol fonts would be installed using one of the '`microsoft-symbol`' and '`apple-roman`' encodings. A number of symbol fonts, however, are not marked as such; such fonts should be installed using '`microsoft-cp1252`', or, for older fonts, '`microsoft-win3.1`'.

In order to guarantee consistent results (especially between Type 1 and TrueType versions of the same font), it is possible to define a special encoding for a given font. This has already been done for the '`ZapfDingbats`' font; see the file '`encodings/adobe-dingbats.enc`'.

3.6 Using badly encoded font files

A number of text fonts are incorrectly encoded. Incorrect encoding is sometimes done by design, in order to make a font for an exotic script appear like an ordinary Western text font. It is often due to the font designer's laziness or incompetence; in particular, most people seem to find it easier to invent idiosyncratic glyph names rather than follow the Adobe glyph list.

There are two ways of dealing with such fonts: using them with the encoding they were designed for, and creating an *ad hoc* encoding file.

Of course, most of the time the proper fix would be to hit the font designer very hard on the head with the PLRM (preferably the first edition, as it was published in hardcover).

3.6.1 Using fonts with the designer's encoding

In the case of Type 1 fonts, the font designer can specify a default encoding; this encoding is requested by using the '`adobe-fontspecific`' encoding in the XLFD name. Sometimes, the font designer omitted to specify a reasonable default encoding; in this case, you should experiment with '`adobe-standard`', '`iso8859-1`', '`microsoft-cp1252`', and '`microsoft-`

win3.1', ('microsoft-symbol' doesn't make sense for Type 1 fonts).

TrueType fonts do not have a default encoding, and use of the Microsoft Symbol encoding yields strange results with text fonts on some (non-X11) platforms. However, most TrueType fonts are designed with either Microsoft or Apple platforms in mind, so one of 'microsoft-cp1252', 'microsoft-win3.1', or 'apple-roman' should yield reasonable results.

3.6.2 Specifying an ad hoc encoding file

It is always possible to define an encoding file to put the glyphs in a font in any desired order. Again, see the 'encodingsadobe-dingbats.enc/' file to see how this is done.

3.6.3 Specifying font aliases

By following the directions above, you will find yourself with a number of fonts with unusual names -- specifying encodings such as 'adobe-fontspecific', 'microsoft-win3.1' *etc*. In order to use these fonts with standard applications, it may be useful to remap them to their proper names.

This is done by writing a 'fonts.alias' file. The format of this file is similar to the format of the 'fonts.dir' file, except that it maps XLFD names to XLFD names. A 'fonts.alias' file might look as follows:

```
1

"-ogonki-alamakota-medium-r-normal--0-0-0-0-p-0-iso8859-2" \

  "-ogonki-alamakota-medium-r-normal--0-0-0-0-p-0-adobe-fontspecific"
```

(both XLFD names on a single line). The syntax of the 'fonts.alias' file is described in the mkfontdir(1) manual page.

4. New font backends

4.1 New TrueType backends

This version of XFree86 comes with two TrueType backends, known as 'freetype' (formerly 'xfsft') and 'X-TrueType' ('X-TT' for short). Those two backends are incompatible, in that only one can be used at any one time. Users are invited to chose whichever backend they find more useful and stick to it.

The 'freetype' backend resides in the module 'freetype'. Before using it, please check that the 'Module' section of your 'XF86Config' file contains a line that reads

```
Load    "freetype"
```

The 'X-TrueType' backend resides in module 'xtt'. In order to use it, replace the line in your 'XF86Config' file that loads the 'freetype' module with a line reading

```
Load    "xtt"
```

Both TrueType backends delay glyph rasterisation to the time at which a glyph is first used. For this reason, they only provide an approximate value for the 'average width' font property. Users are warned not to rely on the average width of a font having an accurate value.

Both backends also support an optimisation for character-cell fonts (fonts with all glyph metrics equal, or terminal fonts). A font with an XLFD specifying a character-cell spacing 'c', as in

```
-misc-mincho-medium-r-normal--0-0-0-0-c-0-jisx0208.1990-0
```

will not rasterise glyphs at metrics computation time, but instead trust the font really to be a character-cell font. Users are encouraged to make use of this optimisation when useful, but be warned that not all monospaced fonts are character-cell fonts.

4.1.1 The ‘freetype’ TrueType backend

The ‘freetype’ backend (formerly ‘xfsft’) is a backend based on the FreeType library (see www.freetype.org) with support for the ‘fontenc’ style of internationalisation (see *The fontenc layer* (section 3.1, page 4)). This backend supports TrueType Font files (`*.ttf`) and TrueType Collections (`*.ttc`).

In order to access the faces in a TrueType Collection file, the face number must be specified in the fonts.dir file before the filename within colons. For example,

```
:2:mincho.ttc -misc-mincho-medium-r-normal--0-0-0-0-c-0-jisx0208.1990-0
```

refers to face 2 in the ‘`mincho.ttc`’ TrueType Collection file.

4.1.2 The ‘X-TrueType’ TrueType backend

The ‘X-TrueType’ backend is another backend based on the FreeType library. X-TrueType doesn’t use the ‘fontenc’ layer for managing font encodings, but instead uses its own database of encodings. However, X-TrueType includes a large number of encodings, and any encoding you need is likely to be present in X-TrueType.

X-TrueType extends the ‘`fonts.dir`’ syntax with a number of options, known as ‘TTCap’. A ‘TTCap’ entry follows the general syntax

```
:option=value:
```

and should be specified before the filename.

The most useful TTCap option is used to specify the face number to use with TTCs; it carries the name ‘`fn`’. This means that face 2 of font file ‘`mincho.ttc`’ is specified using:

```
:fn=2:mincho.ttc -misc-mincho-medium-r-normal--0-0-0-0-c-0-jisx0208.1990-0
```

More information on the TTCap syntax, and on X-TrueType in general, may be found on

```
<URL:http://hawk.ise.chuo-u.ac.jp/student/person/tshiozak/x-tt/index-eng.html>
```

4.2 Support for CID-keyed fonts

The CID-keyed font format was designed by Adobe Systems for fonts with large character sets. It is described in the Adobe Technical Notes nr. 5092, "Overview of the CID-Keyed Font Technology," nr. 5014, "CMap and CIDFont File Format Specification," and others, available from

```
<URL:http://partners.adobe.com/supportservice/devrelations/typeforum/cidfonts.html>
```

Sample CID-keyed fonts can be found at:

```
<URL:ftp://ftp.oreilly.com/pub/examples/nutshell/cjkv/adobe/>
```

Support for CID-keyed fonts in XFree86 is controlled by the two switches ‘`BuildCID`’ and `BuildCIDFonts`. Make sure that those switches are turned on (in the directory `xc/config/cf`) when XFree86 is built. By default, they should be set to `YES`, unless you are building XFree86 for a small memory footprint, in which case they should be set to `NO`.

The CID-keyed font backend does not use the ‘fontenc’ layer, but instead uses the standard ‘CMap’ method of recoding CID-keyed fonts.

4.2.1 Using CID-keyed fonts

As shown in the sample install file `/usr/X11R6/lib/X11/XF86Config.eg`, the font directory CID should be specified as part of the XFree86 font path:

```
FontPath    "/usr/X11R6/lib/X11/fonts/CID/"
```

in the '`XF86Config`' file. When the CID font directory is on the font path it must contain at least the empty files fonts.dir and fonts.scale. Sample '`fonts.dir`' and '`fonts.scale`' files, with 0 entries, are installed by default.

A sample CID-keyed font is provided in the file:

```
test/xsuite/xtest/CID
```

The test directory was given the same name as the CID font directory, because it shows how a CID-keyed font should be installed. It contains a number of subdirectories, and any CID font directory should have the same directory structure.

When installing CID-keyed fonts, the empty fonts.scale and fonts.dir files in the directory:

```
xc/fonts/scaled/CID
```

should be replaced by `fonts.scale` and `fonts.dir` files with a number of entries of the form:

```
1

Adobe-Korea1/Munhwa-Regular--Adobe-Korea1-0.cid \

  -adobe-munhwa-medium-r-normal--0-0-0-0-p-0-adobe.korea1-0
```

(the font file name and the XLFD name should be on the same line). Note that the first column does not specify an actual filename; instead, it specifies the PostScript name of the CID-keyed font, followed by the extension '`.cid`'. The actual names of the files used will be derived from this PostScript name.

CID-keyed fonts are divided in groups by character collection. For example, the Korean font:

```
Munhwa-Regular--Adobe-Korea1-0
```

is in a subdirectory '`Adobe-Korea1`'.

The PostScript name of a CID-keyed font consists of two parts, the *CIDFontName* and the *CMapName*, separated by two dashes. For instance, in the case of the font name

```
Munhwa-Regular--Adobe-Korea1-0
```

the *CIDFontName* is '`Munhwa-Regular`' while the *CMapName* is '`Adobe-Korea1`'.

Each CID-keyed font consist of a CIDFont file and one or more CMap files. The CIDFont file contains the description of each character in a font. It is stored in the subdirectory CIDFont of the Adobe-Korea1 directory. The directory structure looks as following:

```
CID/Adobe-Korea1/CIDFont/Munhwa-Regular

CID/Adobe-Korea1/CMap/Adobe-Korea1-0

CID/Adobe-Korea1/AFM/Munhwa-Regular.afm

CID/Adobe-Korea1/CFM

CID/fonts.dir

CID/fonts.scale
```

The file 'Munhwa-Regular.afm' is an Adobe Font Metric File (AFM). The directory 'CFM' will be used for summaries of that font metric file, which will be computed later.

When the CID-keyed files are installed you can run the utility

```
/usr/X11R6/bin/mkcfm
```

to create the summaries of font metric file (*.cfm), and to put them in appropriate subdirectories. By default, the program works on the directory:

```
/usr/X11R6/lib/X11/fonts/CID
```

A different directory can be specified on the command line of 'mkcfm.'

'mkcfm' should be run as root, as it needs to write its output to a system directory. If the program determines that it cannot write in the designated 'CFM' subdirectories, it will display a message, and switch to current directory.

Unless 'mkcfm' is run, opening large CID-keyed fonts will take a significant amount of time. 'mkcfm' should be run again whenever a change is made to any of the CID-keyed fonts, or when the CID-keyed fonts are copied to a machine with a different architecture.

4.2.2 Limitations

The current version of the CID-keyed fonts backend only supports the CMaps used for horizontal text (*e.g.* the CMap 'KSC-EUC-H' will be used, but not 'KSC-EUC-V'). This limitation is due to the fact that the core X11 protocol only provides support for horizontal writing.

CONTENTS

`$XFree86: xc/programs/Xserver/hw/xfree86/doc/sgml/fonts.sgml,v 1.7 2000/03/06 22:59:25 dawes Exp`