X Logical Font Description Conventions

Version 1.5

X Consortium Standard

X Version 11, Release 6.4

Jim Flowers
Digital Equipment Corporation

Version 1.5 edited by Stephen Gildea
X Consortium, Inc.

1. Introduction

It is a requirement that X client applications must be portable across server implementations, with very different file systems, naming conventions, and font libraries. However, font access requests, as defined by the *X Window System Protocol*, neither specify server-independent conventions for font names nor provide adequate font properties for logically describing typographic fonts.

X clients must be able to dynamically determine the fonts available on any given server so that understandable information can be presented to the user or so that intelligent font fallbacks can be chosen. It is desirable for the most common queries to be accomplished without the overhead of opening each font and inspecting font properties, by means of simple **ListFonts** requests. For example, if a user selected a Helvetica typeface family, a client application should be able to query the server for all Helvetica fonts and present only those setwidths, weights, slants, point sizes, and character sets available for that family.

This document gives a standard logical font description (hereafter referred to as XLFD) and the conventions to be used in the core protocol so that clients can query and access screen type libraries in a consistent manner across all X servers. In addition to completely specifying a given font by means of its **FontName**, the XLFD also provides for a standard set of key **FontProperties** that describe the font in more detail.

The XLFD provides an adequate set of typographic font properties, such as CAP_HEIGHT, X_HEIGHT, and RELATIVE_SETWIDTH, for publishing and other applications to do intelligent font matching or substitution when handling documents created on some foreign server that use potentially unknown fonts. In addition, this information is required by certain clients to position subscripts automatically and determine small capital heights, recommended leading, word-space values, and so on.

2. Requirements and Goals

The XLFD meets the short-term and long-term goals to have a standard logical font description that:

- Provides unique, descriptive font names that support simple pattern matching
- Supports multiple font vendors, arbitrary character sets, and encodings
- Supports naming and instancing of scalable and polymorphic fonts
- Supports transformations and subsetting of fonts
- Is independent of X server and operating or file system implementations
- Supports arbitrarily complex font matching or substitution
- Is extensible

2.1. Provide Unique and Descriptive Font Names

It should be possible to have font names that are long enough and descriptive enough to have a reasonable probability of being unique without inventing a new registration organization. Resolution and size-dependent font masters, multivendor font libraries, and so on must be anticipated and handled by the font name alone.

The name itself should be structured to be amenable to simple pattern matching and parsing, thus allowing X clients to restrict font queries to some subset of all possible fonts in the server.

2.2. Support Multiple Font Vendors and Character Sets

The font name and properties should distinguish between fonts that were supplied by different font vendors but that possibly share the same name. We anticipate a highly competitive font market where users will be able to buy fonts from many sources according to their particular requirements.

A number of font vendors deliver each font with all glyphs designed for that font, where charset mappings are defined by encoding vectors. Some server implementations may force these mappings to proprietary or standard charsets statically in the font data. Others may desire to perform the mapping dynamically in the server. Provisions must be made in the font name that allows a font request to specify or identify specific charset mappings in server environments where multiple charsets are supported.

2.3. Support Scalable and Polymorphic Fonts

If a font source can be scaled to an arbitrary size or varied in other ways, it should be possible for an application to determine that fact from the font name, and the application should be able to construct a font name for any specific instance.

2.4. Support Transformations and Subsetting of Fonts

Arbitrary two-dimensional linear transformations of fonts should be able to be requested by applications. Since such transformed fonts may be used for special effects requiring a few characters from each of many differently transformed fonts, it should be possible to request only a few characters from a font for efficiency.

2.5. Be Independent of X Server and Operating or File System Implementations

X client applications that require a particular font should be able to use the descriptive name without knowledge of the file system or other repository in use by the server. However, it should be possible for servers to translate a given font name into a file name syntax that it knows how to deal with, without compromising the uniqueness of the font name. This algorithm should be reversible (exactly how this translation is done is implementation dependent).

2.6. Support Arbitrarily Complex Font Matching and Substitution

In addition to the font name, the XLFD should define a standard list of descriptive font properties, with agreed-upon fallbacks for all fonts. This allows client applications to derive font-specific formatting or display data and to perform font matching or substitution when asked to handle potentially unknown fonts, as required.

2.7. Be Extensible

The XLFD must be extensible so that new and/or private descriptive font properties can be added to conforming fonts without making existing X client or server implementations obsolete.

3. X Logical Font Description

XLFD is divided into two basic components: the **FontName**, which gives all font information needed to uniquely identify a font in X protocol requests (for example, **OpenFont**, **ListFonts**, and so on) and a variable list of optional **FontProperties**, which describe a font in more detail.

The **FontName** is used in font queries and is returned as data in certain X protocol requests. It is also specified as the data value for the **FONT** item in the X Consortium Character Bitmap Distribution Format Standard (BDF V2.1).

The **FontProperties** are supplied on a font-by-font basis and are returned as data in certain X protocol requests as part of the **XFontStruct** data structure. The names and associated data values for each of the **FontProperties** may also appear as items of the **STARTPROPERTIES**...**ENDPROPERTIES** list in the BDF V2.1 specification.

3.1. FontName

Each **FontName** is logically composed of two strings: a **FontNameRegistry** prefix that is followed by a **FontNameSuffix**. The **FontName** uses the ISO 8859-1 encoding. The **FontNameRegistry** is an x-registered-name (a name that has been registered with the X Consortium) that identifies the registration authority that owns the specified **FontNameSuffix** syntax and semantics.

All font names that conform to this specification are to use a **FontNameRegistry** prefix, which is defined to be the string "−" (HYPHEN). All **FontNameRegistry** prefixes of the form: +*version*−, where the specified version indicates some future XLFD specification, are reserved by the X Consortium for future extensions to XLFD font names. If required, extensions to the current XLFD font name shall be constructed by appending new fields to the current structure, each delimited by the existing field delimiter. The availability of other **FontNameRegistry** prefixes or fonts that support other registries is server implementation dependent.

In the X protocol specification, the **FontName** is required to be a string; hence, numeric field values are represented in the name as string equivalents. All **FontNameSuffix** fields are also defined as **FontProperties**; numeric property values are represented as signed or unsigned integers, as appropriate.

3.1.1. FontName Syntax

The **FontName** is a structured, parsable string (of type STRING8) whose Backus-Naur Form syntax description is as follows:

FontName ::=	XFontNameRegistry XFontNameSuffix | PrivFontNameRegistry PrivFontNameSuffix
XFontNameRegistry ::=	XFNDelim | XFNExtPrefix Version XFNDelim
XFontNameSuffix ::=	FOUNDRY XFNDelim FAMILY_NAME XFNDelim WEIGHT_NAME XFNDelim SLANT XFNDelim SETWIDTH_NAME XFNDelim ADD_STYLE_NAME XFNDelim PIXEL_SIZE XFNDelim POINT_SIZE XFNDelim RESOLUTION_X XFNDelim RESOLUTION_Y XFNDelim SPACING XFNDelim AVERAGE_WIDTH XFNDelim CHARSET_REGISTRY XFNDelim CHARSET_ENCODING
Version ::=	STRING8 − the XLFD version that defines an extension to the font name syntax (for example, "1.4")
XFNExtPrefix ::=	OCTET − "+" (PLUS)
XFNDelim ::=	OCTET − "−" (HYPHEN)
PrivFontNameRegistry ::=	STRING8 − other than those strings reserved by XLFD
PrivFontNameSuffix ::=	STRING8

Field values are constructed as strings of ISO 8859-1 graphic characters, excluding the following:

- "−" (HYPHEN), the XLFD font name delimiter character
- "?" (QUESTION MARK) and "*" (ASTERISK), the X protocol font name wildcard characters
- "," (COMMA), used by Xlib to separate XLFD font names in a font set.
- """ (QUOTATION MARK), used by some commercial products to quote a font name.

Alphabetic case distinctions are allowed but are for human readability concerns only. Conforming X servers will perform matching on font name query or open requests independent of case. The entire font name string must have no more than 255 characters. It is recommended that clients construct font name query patterns by explicitly including all field delimiters to avoid unexpected results. Note that SPACE is a valid character of a **FontName** field; for example, the string “ITC Avant Garde Gothic” might be a FAMILY_NAME.

3.1.2. FontName Field Definitions

This section discusses the **FontName**:

- FOUNDRY field
- FAMILY_NAME field
- WEIGHT_NAME field
- SLANT field
- SETWIDTH_NAME field
- ADD_STYLE_NAME field
- PIXEL_SIZE field
- POINT_SIZE field
- RESOLUTION_X and RESOLUTION_Y fields
- SPACING field
- AVERAGE_WIDTH field
- CHARSET_REGISTRY and CHARSET_ENCODING fields

3.1.2.1. FOUNDRY Field

FOUNDRY is an x-registered-name, the name or identifier of the digital type foundry that digitized and supplied the font data, or if different, the identifier of the organization that last modified the font shape or metric information.

The reason this distinction is necessary is that a given font design may be licensed from one source (for example, ITC) but digitized and sold by any number of different type suppliers. Each digital version of the original design, in general, will be somewhat different in metrics and shape from the idealized original font data, because each font foundry, for better or for worse, has its own standards and practices for tweaking a typeface for a particular generation of output technologies or has its own perception of market needs.

It is up to the type supplier to register with the X Consortium a suitable name for this **FontName** field according to the registration procedures defined by the Consortium.

The X Consortium shall define procedures for registering foundry and other names and shall maintain and publish, as part of its public distribution, a registry of such registered names for use in XLFD font names and properties.

3.1.2.2. FAMILY_NAME Field

FAMILY_NAME is a string that identifies the range or family of typeface designs that are all variations of one basic typographic style. This must be spelled out in full, with words separated by spaces, as required. This name must be human-understandable and suitable for presentation to a font user to identify the typeface family.

It is up to the type supplier to supply and maintain a suitable string for this field and font property, to secure the proper legal title to a given name, and to guard against the infringement of other’s copyrights or trademarks. By convention, FAMILY_NAME is not translated. FAMILY_NAME may include an indication of design ownership if considered a valid part of the typeface family name.

The following are examples of FAMILY_NAME:

- Helvetica
- ITC Avant Garde Gothic
- Times
- Times Roman
- Bitstream Amerigo
- Stone

3.1.2.3. WEIGHT_NAME Field

WEIGHT_NAME is a string that identifies the font's typographic weight, that is, the nominal blackness of the font, according to the FOUNDRY's judgment. This name must be human-understandable and suitable for presentation to a font user. The value "0" is used to indicate a polymorphic font (see section 6).

The interpretation of this field is somewhat problematic because the typographic judgment of weight has traditionally depended on the overall design of the typeface family in question; that is, it is possible that the DemiBold weight of one font could be almost equivalent in typographic feel to a Bold font from another family.

WEIGHT_NAME is captured as an arbitrary string because it is an important part of a font's complete human-understandable name. However, it should not be used for font matching or substitution. For this purpose, X client applications should use the weight-related font properties (RELATIVE_WEIGHT and WEIGHT) that give the coded relative weight and the calculated weight, respectively.

3.1.2.4. SLANT Field

SLANT is a code-string that indicates the overall posture of the typeface design used in the font. The encoding is as follows:

Code	English Translation	Description
"R"	Roman	Upright design
"I"	Italic	Italic design, slanted clockwise from the vertical
"O"	Oblique	Obliqued upright design, slanted clockwise from the vertical
"RI"	Reverse Italic	Italic design, slanted counterclockwise from the vertical
"RO"	Reverse Oblique	Obliqued upright design, slanted counterclockwise from the vertical
"OT"	Other	Other
numeric	Polymorphic	See section 6 on polymorphic font support.

The SLANT codes are for programming convenience only and usually are converted into their equivalent human-understandable form before being presented to a user.

3.1.2.5. SETWIDTH_NAME Field

SETWIDTH_NAME is a string that gives the font's typographic proportionate width, that is, the nominal width per horizontal unit of the font, according to the FOUNDRY's judgment. The value "0" is used to indicate a polymorphic font (see section 6).

As with WEIGHT_NAME, the interpretation of this field or font property is somewhat problematic, because the designer's judgment of setwidth has traditionally depended on the overall design of the typeface family in question. For purposes of font matching or substitution, X client applications should either use the RELATIVE_SETWIDTH font property that gives the relative coded proportionate width or calculate the proportionate width.

The following are examples of SETWIDTH_NAME:

- Normal
- Condensed
- Narrow
- Double Wide

3.1.2.6. ADD_STYLE_NAME Field

ADD_STYLE_NAME is a string that identifies additional typographic style information that is not captured by other fields but is needed to identify the particular font. The character "[" anywhere in the field is used to indicate a polymorphic font (see section 6).

ADD_STYLE_NAME is not a typeface classification field and is only used for uniqueness. Its use, as such, is not limited to typographic style distinctions.

The following are examples of ADD_STYLE_NAME:

- Serif
- Sans Serif
- Informal
- Decorated

3.1.2.7. PIXEL_SIZE Field

PIXEL_SIZE gives the body size of the font at a particular POINT_SIZE and RESOLUTION_Y. PIXEL_SIZE is either an integer-string or a string beginning with "[". A string beginning with "[" represents a matrix (see section 4). PIXEL_SIZE usually incorporates additional vertical spacing that is considered part of the font design. (Note, however, that this value is not necessarily equivalent to the height of the font bounding box.) Zero is used to indicate a scalable font (see section 5).

PIXEL_SIZE usually is used by X client applications that need to query fonts according to device-dependent size, regardless of the point size or vertical resolution for which the font was designed.

3.1.2.8. POINT_SIZE Field

POINT_SIZE gives the body size for which the font was designed. POINT_SIZE is either an integer-string or a string beginning with "[". A string beginning with "[" represents a matrix (see section 4). This field usually incorporates additional vertical spacing that is considered part of the font design. (Note, however, that POINT_SIZE is not necessarily equivalent to the height of the font bounding box.) POINT_SIZE is expressed in decipoints (where points are as defined in the X protocol or 72.27 points equal 1 inch). Zero is used to indicate a scalable font (see section 5).

POINT_SIZE and RESOLUTION_Y are used by X clients to query fonts according to device-independent size to maintain constant text size on the display regardless of the PIXEL_SIZE used for the font.

3.1.2.9. RESOLUTION_X and RESOLUTION_Y Fields

RESOLUTION_X and RESOLUTION_Y are unsigned integer-strings that give the horizontal and vertical resolution, measured in pixels or dots per inch (dpi), for which the font was designed. Zero is used to indicate a scalable font (see section 5). Horizontal and vertical values are required because a separate bitmap font must be designed for displays with very different aspect ratios (for example, 1:1, 4:3, 2:1, and so on).

The separation of pixel or point size and resolution is necessary because X allows for servers with very different video characteristics (for example, horizontal and vertical resolution, screen and pixel size, pixel shape, and so on) to potentially access the same font library. The font name, for example, must differentiate between a 14-point font designed for 75 dpi (body size of about 14 pixels) or a 14-point font designed for 150 dpi (body size of about 28 pixels). Further, in servers that implement some or all fonts as continuously scaled and scan-converted outlines, POINT_SIZE and RESOLUTION_Y will help the server to differentiate between potentially separate font masters for text, title, and display sizes or for other typographic considerations.

3.1.2.10. SPACING Field

SPACING is a code-string that indicates the escapement class of the font, that is, monospace (fixed pitch), proportional (variable pitch), or charcell (a special monospaced font that conforms to the traditional data-processing character cell font model). The encoding is as follows:

Code	English Translation	Description
"P"	Proportional	A font whose logical character widths vary for each glyph. Note that no other restrictions are placed on the metrics of a proportional font.
"M"	Monospaced	A font whose logical character widths are constant (that is, every glyph in the font has the same logical width). No other restrictions are placed on the metrics of a monospaced font.
"C"	CharCell	A monospaced font that follows the standard typewriter character cell model (that is, the glyphs of the font can be modeled by X clients as "boxes" of the same width and height that are imaged side-by-side to form text strings or top-to-bottom to form text lines). By definition, all glyphs have the same logical character width, and no glyphs have "ink" outside of the character cell. There is no kerning (that is, on a per-character basis with positive metrics: 0 <= left-bearing <= right-bearing <= width; with negative metrics: width <= left-bearing <= right-bearing <= zero). Also, the vertical extents of the font do not exceed the vertical spacing (that is, on a per-character basis: ascent <= font-ascent and descent <= font-descent). The cell height = font-descent + font-ascent, and the width = AVERAGE_WIDTH.

3.1.2.11. AVERAGE_WIDTH Field

AVERAGE_WIDTH is an integer-string typographic metric value that gives the unweighted arithmetic mean of the absolute value of the width of each glyph in the font (measured in tenths of pixels), multiplied by -1 if the dominant writing direction for the font is right-to-left. A leading "~" (TILDE) indicates a negative value. For monospaced and character cell fonts, this is the width of all glyphs in the font. Zero is used to indicate a scalable font (see section 5).

3.1.2.12. CHARSET_REGISTRY and CHARSET_ENCODING Fields

The character set used to encode the glyphs of the font (and implicitly the font's glyph repertoire), as maintained by the X Consortium character set registry. CHARSET_REGISTRY is an x-registered-name that identifies the registration authority that owns the specified encoding. CHARSET_ENCODING is a registered name that identifies the coded character set as defined by that registration authority and, optionally, a subsetting hint.

Although the X protocol does not explicitly have any knowledge about character set encodings, it is expected that server implementors will prefer to embed knowledge of certain proprietary or standard charsets into their font library for reasons of performance and convenience. The CHARSET_REGISTRY and CHARSET_ENCODING fields or properties allow an X client font request to specify a specific charset mapping in server environments where multiple charsets are supported. The availability of any particular character set is font and server implementation dependent.

To prevent collisions when defining character set names, it is recommended that CHARSET_REGISTRY and CHARSET_ENCODING name pairs be constructed according to the following conventions:

CharsetRegistry ::=	StdCharsetRegistryName | PrivCharsetRegistryName
CharsetEncoding ::=	StdCharsetEncodingName | PrivCharsetEncodingName
StdCharsetRegistryName ::=	StdOrganizationId StdNumber | StdOrganizationId StdNumber Dot Year
PrivCharsetRegistryName ::=	OrganizationId STRING8
StdCharsetEncodingName ::=	STRING8−numeric part number of referenced standard
PrivCharsetEncodingName ::=	STRING8
StdOrganizationId ::=	STRING8−the registered name or acronym of the referenced standard organization
StdNumber ::=	STRING8−referenced standard number
OrganizationId ::=	STRING8−the registered name or acronym of the organization
Dot ::=	OCTET−"." (FULL STOP)
Year ::=	STRING8−numeric year (for example, 1989)

The X Consortium shall maintain and publish a registry of such character set names for use in X protocol font names and properties as specified in XLFD.

The ISO Latin-1 character set shall be registered by the X Consortium as the CHARSET_REGISTRY-CHARSET_ENCODING value pair: "ISO8859-1".

If the CHARSET_ENCODING contains a "[" (LEFT SQUARE BRACKET), the "[" and the characters after it up to a "]" (RIGHT SQUARE BRACKET) are a subsetting hint telling the font source that the client is interested only in a subset of the characters of the font. The font source can, optionally, return a font that contains only those characters or any superset of those characters. The client can expect to obtain valid glyphs and metrics only for those characters, and not for any other characters in the font. The font properties may optionally be calculated by considering only the characters in the subset.

The BNF for the subsetting hint is

Subset ::=	LeftBracket RangeList RightBracket
RangeList ::=	Range | Range Space RangeList
Range ::=	Number | Number Underscore Number
Number ::=	"0x" HexNumber | DecNumber
HexNumber ::=	HexDigit | HexDigit HexNumber

DecNumber ::=	DecDigit | DecDigit DecNumber
DecDigit ::=	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
HexDigit ::=	DecDigit | "a" | "b" | "c" | "d" | "e" | "f"
LeftBracket ::=	"[" (LEFT SQUARE BRACKET)
RightBracket ::=	"]" (RIGHT SQUARE BRACKET)
Space ::=	" " (SPACE)
Underscore ::=	"_" (LOW LINE)

Each Range specifies characters that are to be part of the subset included in the font. A Range containing two Numbers specifies the first and last character, inclusively, of a range of characters. A Range that is a single Number specifies a single character to be included in the font. A HexNumber is interpreted as a hexadecimal number. A DecNumber is interpreted as a decimal number. The font consists of the union of all the Ranges in the RangeList.

For example,

```
-misc-fixed-medium-r-normal--0-0-0-0-c-0-iso8859-1[65 70 80_90]
```

tells the font source that the client is interested only in characters 65, 70, and 80–90.

3.1.3. Examples

The following examples of font names are derived from the screen fonts shipped with the X Consortium distribution.

Font	X FontName
75-dpi Fonts	
Charter 12 pt	-Bitstream-Charter-Medium-R-Normal--12-120-75-75-P-68-ISO8859-1
Charter Bold 12 pt	-Bitstream-Charter-Bold-R-Normal--12-120-75-75-P-76-ISO8859-1
Charter Bold Italic 12 pt	-Bitstream-Charter-Bold-I-Normal--12-120-75-75-P-75-ISO8859-1
Charter Italic 12 pt	-Bitstream-Charter-Medium-I-Normal--12-120-75-75-P-66-ISO8859-1
Courier 8 pt	-Adobe-Courier-Medium-R-Normal--8-80-75-75-M-50-ISO8859-1
Courier 10 pt	-Adobe-Courier-Medium-R-Normal--10-100-75-75-M-60-ISO8859-1
Courier 12 pt	-Adobe-Courier-Medium-R-Normal--12-120-75-75-M-70-ISO8859-1
Courier 24 pt	-Adobe-Courier-Medium-R-Normal--24-240-75-75-M-150-ISO8859-1
Courier Bold 10 pt	-Adobe-Courier-Bold-R-Normal--10-100-75-75-M-60-ISO8859-1
Courier Bold Oblique 10 pt	-Adobe-Courier-Bold-O-Normal--10-100-75-75-M-60-ISO8859-1
Courier Oblique 10 pt	-Adobe-Courier-Medium-O-Normal--10-100-75-75-M-60-ISO8859-1
100-dpi Fonts	
Symbol 10 pt	-Adobe-Symbol-Medium-R-Normal--14-100-100-100-P-85-Adobe-FONTSPECIFIC
Symbol 14 pt	-Adobe-Symbol-Medium-R-Normal--20-140-100-100-P-107-Adobe-FONTSPECIFIC
Symbol 18 pt	-Adobe-Symbol-Medium-R-Normal--25-180-100-100-P-142-Adobe-FONTSPECIFIC
Symbol 24 pt	-Adobe-Symbol-Medium-R-Normal--34-240-100-100-P-191-Adobe-FONTSPECIFIC
Times Bold 10 pt	-Adobe-Times-Bold-R-Normal--14-100-100-100-P-76-ISO8859-1
Times Bold Italic 10 pt	-Adobe-Times-Bold-I-Normal--14-100-100-100-P-77-ISO8859-1
Times Italic 10 pt	-Adobe-Times-Medium-I-Normal--14-100-100-100-P-73-ISO8859-1
Times Roman 10 pt	-Adobe-Times-Medium-R-Normal--14-100-100-100-P-74-ISO8859-1

3.2. Font Properties

All font properties are optional but will generally include the font name fields and, on a font-by-font basis, any other useful font descriptive and use information that may be required to use the font intelligently. The XLFD specifies an extensive set of standard X font properties, their interpretation, and fallback rules when the property is not defined for a given font. The goal is to provide client applications with enough font information to be able to make automatic formatting and display decisions with good typographic results.

Font property names use the ISO 8859-1 encoding.

Additional standard X font property definitions may be defined in the future and private properties may exist in X fonts at any time. Private font properties should be defined to conform to the general mechanism defined in the X protocol to prevent overlap of name space and ambiguous property names, that is, private font property names are of the form: "_" (LOW LINE), followed by the organizational identifier, followed by "_" (LOW LINE), and terminated with the property name.

The Backus-Naur Form syntax description of X font properties is as follows:

Properties ::=	OptFontPropList
OptFontPropList ::=	NULL | OptFontProp OptFontPropList
OptFontProp ::=	PrivateFontProp | XFontProp
PrivateFontProp ::=	STRING8 | Underscore OrganizationId Underscore STRING8
XFontProp ::=	FOUNDRY | FAMILY_NAME | WEIGHT_NAME | SLANT | SETWIDTH_NAME | ADD_STYLE_NAME | PIXEL_SIZE | POINT_SIZE | RESOLUTION_X | RESOLUTION_Y | SPACING | AVERAGE_WIDTH | CHARSET_REGISTRY | CHARSET_ENCODING | QUAD_WIDTH | RESOLUTION | MIN_SPACE | NORM_SPACE | MAX_SPACE | END_SPACE | SUPERSCRIPT_X | SUPERSCRIPT_Y | SUBSCRIPT_X | SUBSCRIPT_Y | UNDERLINE_POSITION | UNDERLINE_THICKNESS | STRIKEOUT_ASCENT | STRIKEOUT_DESCENT | ITALIC_ANGLE | X_HEIGHT | WEIGHT | FACE_NAME | FULL_NAME | FONT | COPYRIGHT | AVG_CAPITAL_WIDTH | AVG_LOWERCASE_WIDTH | RELATIVE_SETWIDTH | RELATIVE_WEIGHT | CAP_HEIGHT | SUPERSCRIPT_ SIZE | FIGURE_WIDTH | SUBSCRIPT_SIZE | SMALL_CAP_SIZE | NOTICE | DESTINATION | FONT_TYPE | FONT_VERSION | RASTERIZER_NAME | RASTERIZER_VERSION | RAW_ASCENT | RAW_DESCENT | RAW_* | AXIS_NAMES | AXIS_LIMITS | AXIS_TYPES
Underscore ::=	OCTET−"_" (LOW LINE)
OrganizationId ::=	STRING8−the registered name of the organization

3.2.1. FOUNDRY

FOUNDRY is as defined in the **FontName** except that the property type is ATOM.

FOUNDRY cannot be calculated or defaulted if not supplied as a font property.

3.2.2. FAMILY_NAME

FAMILY_NAME is as defined in the **FontName** except that the property type is ATOM.

FAMILY_NAME cannot be calculated or defaulted if not supplied as a font property.

3.2.3. WEIGHT_NAME

WEIGHT_NAME is as defined in the **FontName** except that the property type is ATOM.

WEIGHT_NAME can be defaulted if not supplied as a font property, as follows:

```
if (WEIGHT_NAME undefined) then
  WEIGHT_NAME = ATOM("Medium")
```

3.2.4. SLANT

SLANT is as defined in the **FontName** except that the property type is ATOM.

SLANT can be defaulted if not supplied as a font property, as follows:

```
if (SLANT undefined) then
  SLANT = ATOM("R")
```

3.2.5. SETWIDTH_NAME

SETWIDTH_NAME is as defined in the **FontName** except that the property type is ATOM.

SETWIDTH_NAME can be defaulted if not supplied as a font property, as follows:

```
if (SETWIDTH_NAME undefined) then
  SETWIDTH_NAME = ATOM("Normal")
```

3.2.6. ADD_STYLE_NAME

ADD_STYLE_NAME is as defined in the **FontName** except that the property type is ATOM.

ADD_STYLE_NAME can be defaulted if not supplied as a font property, as follows:

```
if (ADD_STYLE_NAME undefined) then
  ADD_STYLE_NAME = ATOM("")
```

3.2.7. PIXEL_SIZE

PIXEL_SIZE is as defined in the **FontName** except that the property type is INT32.

X clients requiring pixel values for the various typographic fixed spaces (em space, en space, and thin space) can use the following algorithm for computing these values from other properties specified for a font:

```
DeciPointsPerInch = 722.7
EMspace = ROUND ((RESOLUTION_X * POINT_SIZE) / DeciPointsPerInch)
ENspace = ROUND (EMspace / 2)
THINspace = ROUND (EMspace / 3)
```

where a slash (/) denotes real division, an asterisk (*) denotes real multiplication, and ROUND denotes a function that rounds its real argument *a* up or down to the next integer. This rounding is done according to X = FLOOR (*a* + 0.5), where FLOOR is a function that rounds its real argument down to the nearest integer.

PIXEL_SIZE can be approximated if not supplied as a font property, according to the following algorithm:

```
DeciPointsPerInch = 722.7
if (PIXEL_SIZE undefined) then
  PIXEL_SIZE = ROUND ((RESOLUTION_Y * POINT_SIZE) / DeciPointsPerInch)
```

3.2.8. POINT_SIZE

POINT_SIZE is as defined in the **FontName** except that the property type is INT32.

X clients requiring device-independent values for em space, en space, and thin space can use the following algorithm:

```
EMspace = ROUND (POINT_SIZE / 10)
ENspace = ROUND (POINT_SIZE / 20)
THINspace = ROUND (POINT_SIZE / 30)
```

Design POINT_SIZE cannot be calculated or approximated.

3.2.9. RESOLUTION_X

RESOLUTION_X is as defined in the **FontName** except that the property type is CARD32.

RESOLUTION_X cannot be calculated or approximated.

3.2.10. RESOLUTION_Y

RESOLUTION_Y is as defined in the **FontName** except that the property type is CARD32.

RESOLUTION_X cannot be calculated or approximated.

3.2.11. SPACING

SPACING is as defined in the **FontName** except that the property type is ATOM.

SPACING can be calculated if not supplied as a font property, according to the definitions given above for the **FontName**.

3.2.12. AVERAGE_WIDTH

AVERAGE_WIDTH is as defined in the **FontName** except that the property type is INT32.

AVERAGE_WIDTH can be calculated if not provided as a font property, according to the following algorithm:

```
if (AVERAGE_WIDTH undefined) then
  AVERAGE_WIDTH = ROUND (MEAN (ABS (width of each glyph in font)) * 10)
      * (if (dominant writing direction L-to-R) then 1 else −1)
```

where MEAN is a function that returns the arithmetic mean of its arguments.

X clients that require values for the number of characters per inch (pitch) of a monospaced font can use the following algorithm using the AVERAGE_WIDTH and RESOLUTION_X font properties:

```
if (SPACING not proportional) then
  CharPitch = (RESOLUTION_X * 10) / AVERAGE_WIDTH
```

3.2.13. CHARSET_REGISTRY

CHARSET_REGISTRY is as defined in the **FontName** except that the property type is ATOM.

CHARSET_REGISTRY cannot be defaulted if not supplied as a font property.

3.2.14. CHARSET_ENCODING

CHARSET_ENCODING is as defined in the **FontName** except that the property type is ATOM.

CHARSET_ENCODING cannot be defaulted if not supplied as a font property.

3.2.15. MIN_SPACE

MIN_SPACE is an integer value (of type INT32) that gives the recommended minimum word-space value to be used with this font.

MIN_SPACE can be approximated if not provided as a font property, according to the following algorithm:

```
if (MIN_SPACE undefined) then
  MIN_SPACE = ROUND(0.75 * NORM_SPACE)
```

3.2.16. NORM_SPACE

NORM_SPACE is an integer value (of type INT32) that gives the recommended normal word-space value to be used with this font.

NORM_SPACE can be approximated if not provided as a font property, according to the following algorithm:

```
DeciPointsPerInch = 722.7
if (NORM_SPACE undefined) then
  if (SPACE glyph exists) then
    NORM_SPACE = width of SPACE
  else NORM_SPACE = ROUND((0.33 * RESOLUTION_X * POINT_SIZE)/ DeciPointsPerInch)
```

3.2.17. MAX_SPACE

MAX_SPACE is an integer value (of type INT32) that gives the recommended maximum word-space value to be used with this font.

MAX_SPACE can be approximated if not provided as a font property, according to the following algorithm:

```
if (MAX_SPACE undefined) then
  MAX_SPACE = ROUND(1.5 * NORM_SPACE)
```

3.2.18. END_SPACE

END_SPACE is an integer value (of type INT32) that gives the recommended spacing at the end of sentences.

END_SPACE can be approximated if not provided as a font property, according to the following algorithm:

```
if (END_SPACE undefined) then
  END_SPACE = NORM_SPACE
```

3.2.19. AVG_CAPITAL_WIDTH

AVG_CAPITAL_WIDTH is an integer value (of type INT32) that gives the unweighted arithmetic mean of the absolute value of the width of each capital glyph in the font, in tenths of pixels, multiplied by −1 if the dominant writing direction for the font is right-to-left. This property applies to both Latin and non-Latin fonts. For Latin fonts, capitals are the glyphs A through Z. This property is usually used for font matching or substitution.

AVG_CAPITAL_WIDTH can be calculated if not provided as a font property, according to the following algorithm:

```
if (AVG_CAPITAL_WIDTH undefined) then
  if (capitals exist) then
    AVG_CAPITAL_WIDTH = ROUND (MEAN
                (ABS (width of each capital glyph)) * 10)
          * (if (dominant writing direction L-to-R) then 1 else −1)
  else AVG_CAPITAL_WIDTH undefined
```

3.2.20. AVG_LOWERCASE_WIDTH

AVG_LOWERCASE_WIDTH is an integer value (of type INT32) that gives the unweighted arithmetic mean width of the absolute value of the width of each lowercase glyph in the font in tenths of pixels, multiplied by −1 if the dominant writing direction for the font is right-to-left. For Latin fonts, lowercase are the glyphs a through z. This property is usually used for font matching or substitution.

Where appropriate, AVG_LOWERCASE_WIDTH can be approximated if not provided as a font property, according to the following algorithm:

```
if (AVG_LOWERCASE_WIDTH undefined) then
  if (lowercase exists) then
    AVG_LOWERCASE_WIDTH = ROUND (MEAN
                (ABS (width of each lowercase glyph)) * 10)
        * (if (dominant writing direction L-to-R) then 1 else −1)
  else AVG_LOWERCASE_WIDTH undefined
```

3.2.21. QUAD_WIDTH

QUAD_WIDTH is an integer typographic metric (of type INT32) that gives the width of a quad (em) space.

Note

> Because all typographic fixed spaces (em, en, and thin) are constant for a given font size (that is, they do not vary according to setwidth), the use of this font property has been deprecated. X clients that require typographic fixed space values are encouraged to discontinue use of QUAD_WIDTH and compute these values from other font properties (for example, PIXEL_SIZE). X clients that require a font-dependent width value should use either the FIGURE_WIDTH or one of the average character width font properties (AVERAGE_WIDTH, AVG_CAPITAL_WIDTH or AVG_LOWERCASE_WIDTH).

3.2.22. FIGURE_WIDTH

FIGURE_WIDTH is an integer typographic metric (of type INT32) that gives the width of the tabular figures and the dollar sign, if suitable for tabular setting (all widths equal). For Latin fonts, these tabular figures are the Arabic numerals 0 through 9.

FIGURE_WIDTH can be approximated if not supplied as a font property, according to the following algorithm:

```
if (numerals and DOLLAR sign are defined & widths are equal) then
  FIGURE_WIDTH = width of DOLLAR
else FIGURE_WIDTH property undefined
```

3.2.23. SUPERSCRIPT_X

SUPERSCRIPT_X is an integer value (of type INT32) that gives the recommended horizontal offset in pixels from the position point to the X origin of synthetic superscript text. If the current position point is at [X,Y], then superscripts should begin at [X + SUPERSCRIPT_X, Y − SUPERSCRIPT_Y].

SUPERSCRIPT_X can be approximated if not provided as a font property, according to the following algorithm:

```
if (SUPERSCRIPT_X undefined) then
  if (TANGENT(ITALIC_ANGLE) defined) then
    SUPERSCRIPT_X = ROUND((0.40 * CAP_HEIGHT) / TANGENT(ITALIC_ANGLE))
  else SUPERSCRIPT_X = ROUND(0.40 * CAP_HEIGHT)
```

where TANGENT is a trigonometric function that returns the tangent of its argument, which is in 1/64 degrees.

3.2.24. SUPERSCRIPT_Y

SUPERSCRIPT_Y is an integer value (of type INT32) that gives the recommended vertical offset in pixels from the position point to the Y origin of synthetic superscript text. If the current position point is at [X,Y], then superscripts should begin at [X + SUPERSCRIPT_X, Y − SUPERSCRIPT_Y].

SUPERSCRIPT_Y can be approximated if not provided as a font property, according to the following algorithm:

```
if (SUPERSCRIPT_Y undefined) then
  SUPERSCRIPT_Y = ROUND(0.40 * CAP_HEIGHT)
```

3.2.25. SUBSCRIPT_X

SUBSCRIPT_X is an integer value (of type INT32) that gives the recommended horizontal offset in pixels from the position point to the X origin of synthetic subscript text. If the current position point is at [X,Y], then subscripts should begin at [X + SUBSCRIPT_X, Y + SUBSCRIPT_Y].

SUBSCRIPT_X can be approximated if not provided as a font property, according to the following algorithm:

```
if (SUBSCRIPT_X undefined) then
  if (TANGENT(ITALIC_ANGLE) defined) then
    SUBSCRIPT_X = ROUND((0.40 * CAP_HEIGHT) / TANGENT(ITALIC_ANGLE))
  else SUBSCRIPT_X = ROUND(0.40 * CAP_HEIGHT)
```

3.2.26. SUBSCRIPT_Y

SUBSCRIPT_Y is an integer value (of type INT32) that gives the recommended vertical offset in pixels from the position point to the Y origin of synthetic subscript text. If the current position point is at [X,Y], then subscripts should begin at [X + SUBSCRIPT_X, Y + SUBSCRIPT_Y].

SUBSCRIPT_Y can be approximated if not provided as a font property, according to the following algorithm:

```
if (SUBSCRIPT_Y undefined) then
  SUBSCRIPT_Y = ROUND(0.40 * CAP_HEIGHT)
```

3.2.27. SUPERSCRIPT_SIZE

SUPERSCRIPT_SIZE is an integer value (of type INT32) that gives the recommended body size of synthetic superscripts to be used with this font, in pixels. This will generally be smaller than the size of the current font; that is, superscripts are imaged from a smaller font offset according to SUPERSCRIPT_X and SUPERSCRIPT_Y.

SUPERSCRIPT_SIZE can be approximated if not provided as a font property, according to the following algorithm:

```
if (SUPERSCRIPT_SIZE undefined) then
  SUPERSCRIPT_SIZE = ROUND(0.60 * PIXEL_SIZE)
```

3.2.28. SUBSCRIPT_SIZE

SUBSCRIPT_SIZE is an integer value (of type INT32) that gives the recommended body size of synthetic subscripts to be used with this font, in pixels. As with SUPERSCRIPT_SIZE, this will generally be smaller than the size of the current font; that is, subscripts are imaged from a smaller font offset according to SUBSCRIPT_X and SUBSCRIPT_Y.

SUBSCRIPT_SIZE can be approximated if not provided as a font property, according to the algorithm:

```
if (SUBSCRIPT_SIZE undefined) then
  SUBSCRIPT_SIZE = ROUND(0.60 * PIXEL_SIZE)
```

3.2.29. SMALL_CAP_SIZE

SMALL_CAP_SIZE is an integer value (of type INT32) that gives the recommended body size of synthetic small capitals to be used with this font, in pixels. Small capitals are generally imaged from a smaller font of slightly more weight. No offset [X,Y] is necessary.

SMALL_CAP_SIZE can be approximated if not provided as a font property, according to the following algorithm:

```
if (SMALL_CAP_SIZE undefined) then
  SMALL_CAP_SIZE = ROUND(PIXEL_SIZE * ((X_HEIGHT
                   + ((CAP_HEIGHT − X_HEIGHT) / 3)) / CAP_HEIGHT))
```

3.2.30. UNDERLINE_POSITION

UNDERLINE_POSITION is an integer value (of type INT32) that gives the recommended vertical offset in pixels from the baseline to the top of the underline. If the current position point is at [X,Y], the top of the baseline is given by [X, Y + UNDERLINE_POSITION].

UNDERLINE_POSITION can be approximated if not provided as a font property, according to the following algorithm:

```
if (UNDERLINE_POSITION undefined) then
  UNDERLINE_POSITION = ROUND((maximum descent) / 2)
```

where *maximum descent* is the maximum descent (below the baseline) in pixels of any glyph in the font.

3.2.31. UNDERLINE_THICKNESS

UNDERLINE_THICKNESS is an integer value (of type INT32) that gives the recommended underline thickness, in pixels.

UNDERLINE_THICKNESS can be approximated if not provided as a font property, according to the following algorithm:

```
CapStemWidth = average width of the stems of capitals
if (UNDERLINE_THICKNESS undefined) then
  UNDERLINE_THICKNESS = CapStemWidth
```

3.2.32. STRIKEOUT_ASCENT

STRIKEOUT_ASCENT is an integer value (of type INT32) that gives the vertical ascent for boxing or voiding glyphs in this font. If the current position is at [X,Y] and the string extent is EXTENT, the upper-left corner of the strikeout box is at [X, Y − STRIKEOUT_ASCENT] and the lower-right corner of the box is at [X + EXTENT, Y + STRIKEOUT_DESCENT].

STRIKEOUT_ASCENT can be approximated if not provided as a font property, according to the following algorithm:

```
if (STRIKEOUT_ASCENT undefined)
  STRIKEOUT_ASCENT = maximum ascent
```

where *maximum ascent* is the maximum ascent (above the baseline) in pixels of any glyph in the font.

3.2.33. STRIKEOUT_DESCENT

STRIKEOUT_DESCENT is an integer value (of type INT32) that gives the vertical descent for boxing or voiding glyphs in this font. If the current position is at [X,Y] and the string extent is EXTENT, the upper-left corner of the strikeout box is at [X, Y − STRIKEOUT_ASCENT] and the lower-right corner of the box is at [X + EXTENT, Y + STRIKEOUT_DESCENT].

STRIKEOUT_DESCENT can be approximated if not provided as a font property, according to the following algorithm:

```
if (STRIKEOUT_DESCENT undefined)
  STRIKEOUT_DESCENT = maximum descent
```

where *maximum descent* is the maximum descent (below the baseline) in pixels of any glyph in the font.

3.2.34. ITALIC_ANGLE

ITALIC_ANGLE is an integer value (of type INT32) that gives the nominal posture angle of the typeface design, in 1/64 degrees, measured from the glyph origin counterclockwise from the three o'clock position.

ITALIC_ANGLE can be defaulted if not provided as a font property, according to the following algorithm:

```
if (ITALIC_ANGLE undefined) then
  ITALIC_ANGLE = (90 * 64)
```

3.2.35. CAP_HEIGHT

CAP_HEIGHT is an integer value (of type INT32) that gives the nominal height of the capital letters contained in the font, as specified by the FOUNDRY or typeface designer.

Certain clients require CAP_HEIGHT to compute scale factors and positioning offsets for synthesized glyphs where this information or designed glyphs are not explicitly provided by the font (for example, small capitals, superiors, inferiors, and so on). CAP_HEIGHT is also a critical factor in font matching and substitution.

CAP_HEIGHT can be approximated if not provided as a font property, according to the following algorithm:

```
if (CAP_HEIGHT undefined) then
  if (Latin font) then
    CAP_HEIGHT = XCharStruct.ascent[glyph X]
  else if (capitals exist) then
     CAP_HEIGHT = XCharStruct.ascent[some unaccented capital glyph]
  else CAP_HEIGHT undefined
```

3.2.36. X_HEIGHT

X_HEIGHT is an integer value (of type INT32) that gives the nominal height above the baseline of the lowercase glyphs contained in the font, as specified by the FOUNDRY or typeface designer.

As with CAP_HEIGHT, X_HEIGHT is required by certain clients to compute scale factors for synthesized small capitals where this information is not explicitly provided by the font resource. X_HEIGHT is a critical factor in font matching and substitution.

X_HEIGHT can be approximated if not provided as a font property, according to the following algorithm:

```
if (X_HEIGHT undefined) then
  if (Latin font) then
    X_HEIGHT = XCharStruct.ascent[glyph x]
  else if (lowercase exists) then
     X_HEIGHT = XCharStruct.ascent[some unaccented lc glyph without an ascender]
  else X_HEIGHT undefined
```

3.2.37. RELATIVE_SETWIDTH

RELATIVE_SETWIDTH is an unsigned integer value (of type CARD32) that gives the coded proportionate width of the font, relative to all known fonts of the same typeface family, according to the type designer's or FOUNDRY's judgment.

RELATIVE_SETWIDTH ranges from 10 to 90 or is 0 if undefined or unknown. The following reference values are defined:

Code	English Translation	Description
0	Undefined	Undefined or unknown
10	UltraCondensed	The lowest ratio of average width to height
20	ExtraCondensed	
30	Condensed	Condensed, Narrow, Compressed, ...
40	SemiCondensed	
50	Medium	Medium, Normal, Regular, ...
60	SemiExpanded	SemiExpanded, DemiExpanded, ...
70	Expanded	
80	ExtraExpanded	ExtraExpanded, Wide, ...
90	UltraExpanded	The highest ratio of average width to height

RELATIVE_SETWIDTH can be defaulted if not provided as a font property, according to the following algorithm:

```
if (RELATIVE_SETWIDTH undefined) then
  RELATIVE_SETWIDTH = 50
```

For polymorphic fonts, RELATIVE_SETWIDTH is not necessarily a linear function of the font's setwidth axis.

X clients that want to obtain a calculated proportionate width of the font (that is, a font-independent way of identifying the proportionate width across all fonts and all font vendors) can use the following algorithm:

```
SETWIDTH = AVG_CAPITAL_WIDTH / (CAP_HEIGHT * 10)
```

where SETWIDTH is a real number with zero being the narrowest calculated setwidth.

3.2.38. RELATIVE_WEIGHT

RELATIVE_WEIGHT is an unsigned integer value (of type CARD32) that gives the coded weight of the font, relative to all known fonts of the same typeface family, according to the type designer's or FOUNDRY's judgment.

RELATIVE_WEIGHT ranges from 10 to 90 or is 0 if undefined or unknown. The following reference values are defined:

Code	English Translation	Description
0	Undefined	Undefined or unknown
10	UltraLight	The lowest ratio of stem width to height
20	ExtraLight	
30	Light	
40	SemiLight	SemiLight, Book, ...
50	Medium	Medium, Normal, Regular,...
60	SemiBold	SemiBold, DemiBold, ...
70	Bold	
80	ExtraBold	ExtraBold, Heavy, ...
90	UltraBold	UltraBold, Black, ..., the highest ratio of stem width to height

RELATIVE_WEIGHT can be defaulted if not provided as a font property, according to the following algorithm:

```
if (RELATIVE_WEIGHT undefined) then
  RELATIVE_WEIGHT = 50
```

For polymorphic fonts, RELATIVE_WEIGHT is not necessarily a linear function of the font's weight axis.

3.2.39. WEIGHT

Calculated WEIGHT is an unsigned integer value (of type CARD32) that gives the calculated weight of the font, computed as the ratio of capital stem width to CAP_HEIGHT, in the range 0 to 1000, where 0 is the lightest weight.

WEIGHT can be calculated if not supplied as a font property, according to the following algorithm:

```
CapStemWidth = average width of the stems of capitals
if (WEIGHT undefined) then
  WEIGHT = ROUND ((CapStemWidth * 1000) / CAP_HEIGHT)
```

A calculated value for weight is necessary when matching fonts from different families because both the RELATIVE_WEIGHT and the WEIGHT_NAME are assigned by the typeface supplier, according to its tradition and practice, and therefore, are somewhat subjective. Calculated WEIGHT provides a font-independent way of identifying the weight across all fonts and all font vendors.

3.2.40. RESOLUTION

RESOLUTION is an integer value (of type INT32) that gives the resolution for which this font was created, measured in 1/100 pixels per point.

Note

> As independent horizontal and vertical design resolution components are required to accommodate displays with nonsquare aspect ratios, the use of this font property has been deprecated, and independent RESOLUTION_X and RESOLUTION_Y font name fields/properties have been defined (see sections 3.1.2.9 and 3.1.2.10). X clients are encouraged to discontinue use of the RESOLUTION property and are encouraged to use the appropriate X,Y resolution properties, as required.

3.2.41. FONT

FONT is a string (of type ATOM) that gives the full XLFD name of the font—that is, the value can be used to open another instance of the same font.

If not provided, the FONT property cannot be calculated.

3.2.42. FACE_NAME

FACE_NAME is a human-understandable string (of type ATOM) that gives the full device-independent typeface name, including the owner, weight, slant, set, and so on but not the resolution, size, and so on. This property may be used as feedback during font selection.

FACE_NAME cannot be calculated or approximated if not provided as a font property.

3.2.43. FULL_NAME

FULL_NAME is the same as FACE_NAME. Its use is deprecated, but it is found on some old fonts.

3.2.44. COPYRIGHT

COPYRIGHT is a human-understandable string (of type ATOM) that gives the copyright information of the legal owner of the digital font data.

This information is a required component of a font but is independent of the particular format used to represent it (that is, it cannot be captured as a comment that could later be thrown away for efficiency reasons).

COPYRIGHT cannot be calculated or approximated if not provided as a font property.

3.2.45. NOTICE

NOTICE is a human-understandable string (of type ATOM) that gives the copyright information of the legal owner of the font design or, if not applicable, the trademark information for the typeface FAMILY_NAME.

Typeface design and trademark protection laws vary from country to country, the USA having no design copyright protection currently while various countries in Europe offer both design and typeface family name trademark protection. As with COPYRIGHT, this information is a required component of a font but is independent of the particular format used to represent it.

NOTICE cannot be calculated or approximated if not provided as a font property.

3.2.46. DESTINATION

DESTINATION is an unsigned integer code (of type CARD32) that gives the font design destination, that is, whether it was designed as a screen proofing font to match printer font glyph widths (WYSIWYG), as an optimal video font (possibly with corresponding printer font) for extended screen viewing (video text), and so on.

The font design considerations are very different, and at current display resolutions, the readability and legibility of these two kinds of screen fonts are very different. DESTINATION allows publishing clients that use X to model the printed page and video text clients, such as on-line documentation browsers, to query for X screen fonts that suit their particular requirements.

The encoding is as follows:

Code	English Translation	Description
0	WYSIWYG	The font is optimized to match the typographic design and metrics of an equivalent printer font.
1	Video text	The font is optimized for screen legibility and readability.

3.2.47. FONT_TYPE

FONT_TYPE is a human-understandable string (of type ATOM) that describes the format of the font data as they are read from permanent storage by the current font source. It is a static attribute of the source data. It can be used by clients to select a type of bitmap or outline font without regard to the rasterizer used to render the font.

Predefined values are as follows:

Value	When applicable
"Bitmap"	Hand-tuned bitmap fonts. Some attempt has been made to optimize the visual appearance of the font for the requested size and resolution.
"Prebuilt"	All bitmap format fonts that cannot be described as "Bitmap", that is, hand-tuned. For example, a bitmap format font that was generated mechanically using a scalable font rasterizer would be considered "Prebuilt", not "Bitmap".
"Type 1"	Any Type 1 font.
"TrueType"	Any TrueType font.
"Speedo"	Any Speedo font.
"F3"	Any F3 font.

Other values may be registered with the X Consortium.

3.2.48. FONT_VERSION

FONT_VERSION is a human-understandable string (of type ATOM) that describes the formal or informal version of the font. **None** is a valid value.

3.2.49. RASTERIZER_NAME

RASTERIZER_NAME is a human-understandable string (of type ATOM) that is the specific name of the rasterizer that has performed some rasterization operation (such as scaling from outlines) on this font.

To define a RASTERIZER_NAME, the following format is recommended:

RasterizerName ::= OrganizationId Space Rasterizer

OrganizationId ::=	STRING8—the X Registry ORGANIZATION name of the rasterizer implementor or maintainer.
Rasterizer ::=	the case-sensitive, human-understandable product name of the rasterizer. Words within this name should be separated by a single SPACE.
Space ::=	OCTET−“ ” (SPACE)

Examples:

X Consortium Bit Scaler
X Consortium Type 1 Rasterizer
X Consortium Speedo Rasterizer
Adobe Type Manager
Sun TypeScaler

If RASTERIZER_NAME is not defined, or is **None**, no rasterization operation has been applied to the FONT_TYPE.

3.2.50. RASTERIZER_VERSION

RASTERIZER_VERSION is a human-understandable string (of type ATOM) that represents the formal or informal version of a font rasterizer. The RASTERIZER_VERSION should match the corresponding product version number known to users, when applicable.

3.2.51. RAW_ASCENT

For a font with a transformation matrix, RAW_ASCENT is the font ascent in 1000 pixel metrics (see section 4.1).

3.2.52. RAW_DESCENT

For a font with a transformation matrix, RAW_DESCENT is the font descent in 1000 pixel metrics (see section 4.1).

3.2.53. RAW_*

For a font with a transformation matrix, all font properties that represent horizontal or vertical sizes or displacements will be accompanied by a new property, named as the original except prefixed with “RAW_”, that is computed as described in section 4.1.

3.2.54. AXIS_NAMES

AXIS_NAMES is a list of all the names of the axes for a polymorphic font, separated by a null (0) byte. These names are suitable for presentation in a user interface (see section 6).

3.2.55. AXIS_LIMITS

AXIS_LIMITS is a list of integers, two for each axis, giving the minimum and maximum allowable values for that axis of a polymorphic font (see section 6).

3.2.56. AXIS_TYPES

AXIS_TYPES is like AXIS_NAMES, but can be registered as having specific semantics (see section 6).

3.3. Built-in Font Property Atoms

The following font property atom definitions were predefined in the initial version of the core protocol:

Font Property/Atom Name	Property Type
MIN_SPACE	INT32
NORM_SPACE	INT32
MAX_SPACE	INT32
END_SPACE	INT32
SUPERSCRIPT_X	INT32
SUPERSCRIPT_Y	INT32
SUBSCRIPT_X	INT32
SUBSCRIPT_Y	INT32
UNDERLINE_POSITION	INT32
UNDERLINE_THICKNESS	INT32
STRIKEOUT_ASCENT	INT32
STRIKEOUT_DESCENT	INT32
FONT_ASCENT	INT32
FONT_DESCENT	INT32
ITALIC_ANGLE	INT32
X_HEIGHT	INT32
QUAD_WIDTH	INT32 − deprecated
WEIGHT	CARD32
POINT_SIZE	INT32
RESOLUTION	CARD32 − deprecated
COPYRIGHT	ATOM
FULL_NAME	ATOM − deprecated
FAMILY_NAME	ATOM
DEFAULT_CHAR	CARD32

4. Matrix Transformations

An XLFD name presented to the server can have the POINT_SIZE or PIXEL_SIZE field begin with the character “[”. If the first character of the field is “[”, the character must be followed with ASCII representations of four floating point numbers and a trailing “]”, with white space separating the numbers and optional white space separating the numbers from the “[” and “]” characters. Numbers use standard floating point syntax but use the character “~” to represent a minus sign in the mantissa or exponent.

The BNF for a matrix transformation string is as follows:

```
MatrixString ::=  LeftBracket OptionalSpace Float Space Float Space
                  Float Space Float OptionalSpace RightBracket
OptionalSpace ::= “” | Space
        Space ::= SpaceChar | SpaceChar Space
        Float ::= Mantissa | Mantissa Exponent
     Mantissa ::= Sign Number | Number
         Sign ::= Plus | Tilde
       Number ::= Integer | Integer Dot Integer | Dot Integer
      Integer ::= Digit | Digit Integer
        Digit ::= “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”
     Exponent ::= “e” SignedInteger | “E” SignedInteger
```

SignedInteger ::=	Sign Integer | Integer
LeftBracket ::=	OCTET – "[" (LEFT SQUARE BRACKET)
RightBracket ::=	OCTET – "]" (RIGHT SQUARE BRACKET)
SpaceChar ::=	OCTET – " " (SPACE)
Tilde ::=	OCTET – "~" (TILDE)
Plus ::=	OCTET – "+" (PLUS)
Dot ::=	OCTET – "." (FULL STOP)

The string ‘‘[a b c d]’’ represents a graphical transformation of the glyphs in the font by the matrix

```
[    a    b    0    ]
[    c    d    0    ]
[    0    0    1    ]
```

All transformations occur around the origin of the glyph. The relationship between the current scalar values and the matrix transformation values is that the scalar value ‘‘N’’ in the POINT_SIZE field produces the same glyphs as the matrix ‘‘[N/10 0 0 N/10]’’ in that field, and the scalar value ‘‘N’’ in the PIXEL_SIZE field produces the same glyphs as the matrix ‘‘[N*RESOLUTION_X/RESOLUTION_Y 0 0 N]’’ in that field.

If matrices are specified for both the POINT_SIZE and PIXEL_SIZE, they must bear the following relationship to each other within an implementation-specific tolerance:

PIXEL_SIZE_MATRIX = [Sx 0 0 Sy] * POINT_SIZE_MATRIX

where

Sx = RESOLUTION_X / 72.27
Sy = RESOLUTION_Y / 72.27

If either the POINT_SIZE or PIXEL_SIZE field is unspecified (either ‘‘0’’ or wildcarded), the preceding formulas can be used to compute one from the other.

4.1. Metrics and Font Properties

In this section, the phrase ‘‘1000 pixel metrics’’ means the metrics that would be obtained if the rasterizer took the base untransformed design used to generate the transformed font and scaled it linearly to a height of 1000 pixels, with no rotation component. Note that there may be no way for the application to actually request this font since the rasterizer may use different outlines or rasterization techniques at that size from the ones used to generate the transformed font.

Notes on properties and metrics:

The per-char ink metrics (lbearing, rbearing, ascent, and descent) represent the ink extent of the transformed glyph around its origin.

The per-char width is the x component of the transformed character width.

The font ascent and descent are the y component of the transformed font ascent or descent.

The FONT property returns a name reflecting the matrix being used—that is, the name returned can be used to open another instance of the same font. The returned name is not necessarily an exact copy of the requested name. If, for example, the user requests

```
-misc-fixed-medium-r-normal--0-[2e1 0 0.0 +10.0]-72-72-c-0-iso8859-1
```

the resulting FONT property might be

```
-misc-fixed-medium-r-normal--[19.9 0 0 10]-[20 0 0 10]-72-72-c-0-iso8859-1
```

The FONT property will always include matrices in both the PIXEL_SIZE and the POINT_SIZE fields.

To allow accurate client positioning of transformed characters, the attributes field of the XCharInfo contains the width of the character in 1000 pixel metrics. This attributes field should be interpreted as a signed integer.

There will always be 2 new font properties defined, RAW_ASCENT and RAW_DESCENT, that hold the ascent and descent in 1000 pixel metrics.

All font properties that represent horizontal widths or displacements have as their value the x component of the transformed width or displacement. All font properties that represent vertical heights or displacements have as their value the y component of the transformed height or displacement. Each such property will be accompanied by a new property, named as the original except prefixed with “RAW_”, that gives the value of the width, height, or displacement in 1000 pixel metrics.

5. Scalable Fonts

The XLFD is designed to support scalable fonts. A scalable font is a font source from which instances of arbitrary size can be derived. A scalable font source might be one or more outlines together with zero or more hand-tuned bitmap fonts at specific sizes and resolutions, or it might be a programmatic description together with zero or more bitmap fonts, or some other format (perhaps even just a single bitmap font).

The following definitions are useful for discussing scalable fonts:

Well-formed XLFD pattern

A pattern string containing 14 hyphens, one of which is the first character of the pattern. Wildcard characters are permitted in the fields of a well-formed XLFD pattern.

Scalable font name

A well-formed XLFD pattern containing no wildcards and containing the digit “0” in the PIXEL_SIZE, POINT_SIZE, and AVERAGE_WIDTH fields.

Scalable fields

The XLFD fields PIXEL_SIZE, POINT_SIZE, RESOLUTION_X, RESOLUTION_Y, and AVERAGE_WIDTH.

Derived instance

The result of replacing the scalable fields of a font name with values to yield a font name that could actually be produced from the font source. A scaling engine is permitted, but not required, to interpret the scalable fields in font names to support anamorphic scaling.

Global list

The list of names that would be returned by an X server for a **ListFonts** protocol request on the pattern “*” if there were no protocol restrictions on the total number of names returned.

The global list consists of font names derived from font sources. If a single font source can support multiple character sets (specified in the CHARSET_REGISTRY and CHARSET_ENCODING fields), each such character set should be used to form a separate font name in the list. For a nonscalable font source, the simple font name for each character set is included in the global list. For a scalable font source, a scalable font name for each character set is included in the list. In addition to the scalable font name, specific derived instance names may also be included in the list. The relative order of derived instances with respect to the scalable font name is not constrained. Finally, font name aliases may also be included in the list. The relative order of aliases with respect to the real font name is not constrained.

The values of the RESOLUTION_X and RESOLUTION_Y fields of a scalable font name are implementation dependent, but to maximize backward compatibility, they should be reasonable nonzero values, for example, a resolution close to that provided by the screen (in a single-screen server). Because some existing applications rely on seeing a collection of point and pixel sizes, server vendors are strongly encouraged in the near term to provide a mechanism for including, for each scalable font name, a set of specific derived instance names. For font sources that contain a collection of hand-tuned bitmap fonts, including names of these instances in the global list is recommended and sufficient.

The X protocol request **OpenFont** on a scalable font name returns a font corresponding to an implementation-dependent derived instance of that font name.

The X protocol request **ListFonts** on a well-formed XLFD pattern returns the following. Starting with the global list, if the actual pattern argument has values containing no wildcards in scalable fields, then substitute each such field into the corresponding field in each scalable font name in the list. For each resulting font name, if the remaining scalable fields cannot be replaced with values to produce a derived instance, remove the font name from the list. Now take the modified list, and perform a simple pattern match against the pattern argument. **ListFonts** returns the resulting list.

For example, given the global list:

```
-Linotype-Times-Bold-I-Normal--0-0-100-100-P-0-ISO8859-1
-Linotype-Times-Bold-R-Normal--0-0-100-100-P-0-ISO8859-1
-Linotype-Times-Medium-I-Normal--0-0-100-100-P-0-ISO8859-1
-Linotype-Times-Medium-R-Normal--0-0-100-100-P-0-ISO8859-1
```

a **ListFonts** request with the pattern:

```
-*-Times-*-R-Normal--*-120-100-100-P-*-ISO8859-1
```

would return:

```
-Linotype-Times-Bold-R-Normal--0-120-100-100-P-0-ISO8859-1
-Linotype-Times-Medium-R-Normal--0-120-100-100-P-0-ISO8859-1
```

ListFonts on a pattern containing wildcards that is not a well-formed XLFD pattern is only required to return the list obtained by performing a simple pattern match against the global list. X servers are permitted, but not required, to use a more sophisticated matching algorithm.

6. Polymorphic Fonts

Fonts that can be varied in ways other than size or resolution are called *polymorphic fonts*. Multiple Master Type 1 font programs are one type of a polymorphic font. Current examples of axes along which the fonts can be varied are width, weight, and optical size; others might include formality or x-height.

To support polymorphic fonts, special values indicating variability are defined for the following XLFD fields:

    WEIGHT_NAME
    SLANT
    SETWIDTH_NAME
    ADD_STYLE_NAME

The string “0” is the special polymorphic value. In the WEIGHT_NAME, SLANT, or SETWIDTH_NAME field, “0” must be the entire field. There may be multiple polymorphic values in the ADD_STYLE_NAME field. They are surrounded by “[” and “]” and separated by a Space, as “[0 0]”. The polymorphic values may coexist with other data in the field. It is recommended that the polymorphic values be at the end of the ADD_STYLE_NAME field.

The font-matching algorithms for a font with polymorphic fields are identical to the matching algorithms for a font with scalable fields.

There are three new font properties to describe the axes of variation, AXIS_NAMES, AXIS_LIMITS, and AXIS_TYPES. AXIS_NAMES is a list of all the names of the axes for the font, separated by a null (0) byte. These names are suitable for presentation in a user interface. AXIS_LIMITS is a list of integers, two for each axis, giving the minimum and maximum allowable values for that axis. AXIS_TYPES is like AXIS_NAMES, but can be registered as having specific semantics.

The axes are listed in the properties in the same order as they appear in the font name. They are matched with font name fields by looking for the special polymorphic values in the font name.

Examples:

The Adobe Myriad MM font program has width and weight axes. Weight can vary from 215 to 830, and width from 300 to 700.

Name:
    `-Adobe-Myriad MM-0-R-0--0-0-0-0-P-0-ISO8859-1`

AXIS_NAMES:
    Weight, Width

AXIS_LIMITS:
    215, 830, 300, 700

AXIS_TYPES:
    Adobe-Weight, Adobe-Width

Sample derived instance:
    `-Adobe-Myriad MM-412-R-575--*-120-100-100-P-*-ISO8859-1`

The Adobe Minion MM Italic font program has width, weight, and optical size axes.

Name:
    `-Adobe-Minion MM-0-I-0-[0]-0-0-0-0-P-0-ISO8859-1`

AXIS_NAMES:
    Weight, Width, Optical size

AXIS_LIMITS:
    345, 620, 450, 600, 6, 72

AXIS_TYPES:
    Adobe-Weight, Adobe-Width, Adobe-OpticalSize

Sample derived instance:
    `-Adobe-Minion MM-550-I-480-[18]-*-180-100-100-P-*-ISO8859-1`

The Adobe Minion MM Swash Italic font program has the same axes and values. This shows how “[0]” in the ADD_STYLE_NAME field can coexist with other words.

Name:
    `-Adobe-Minion MM-0-I-0-Swash[0]-0-0-0-0-P-0-ISO8859-1`

AXIS_NAMES:
    Weight, Width, Optical size

AXIS_LIMITS:
    345, 620, 450, 600, 6, 72

AXIS_TYPES:

Adobe-Weight, Adobe-Width, Adobe-OpticalSize

Sample derived instance:

```
-Adobe-Minion MM-550-I-480-Swash[18]-*-180-100-100-P-*-ISO8859-1
```

The XYZ Abc font, a hypothetical font, has optical size and x-height axes. This shows how there can be more than one polymorphic value in the ADD_STYLE_NAME field.

Name:

```
-XYZ-Abc-Medium-R-Normal-[0 0]-0-0-0-0-P-0-ISO8859-1
```

AXIS_NAMES:

Optical size, X-height

AXIS_LIMITS:

6, 72, 400, 600

AXIS_TYPES:

XYZ-OpticalSize, XYZ-Xheight

Sample derived instance:

```
-XYZ-Abc-Medium-R-Normal-[14 510]-*-140-100-100-P-*-ISO8859-1
```

If an axis allows negative values, a client requests a negative value by using "~" (TILDE) as a minus sign.

Axis types can be registered with the X Consortium, along with their semantics.

If a font name that contains the polymorphic value or a wildcard in a polymorphic field is presented to a font source, the font source is free to substitute any value that is convenient. However, font sources should try to use a value that would be considered *normal* or *medium* for the particular font. For example, if an optical size variable is unresolved, the font source should provide a value appropriate to the size of the font.

The result of specifying an out-of-range value for a polymorphic field is undefined. The font source may treat this as a **BadName** error, treat the value as if it were the closest legal value, or extrapolate to try to accommodate the value.

7. Affected Elements of Xlib and the X Protocol

The following X protocol requests must support the XLFD conventions:

- **OpenFont** – for the name argument
- **ListFonts** – for the pattern argument
- **ListFontsWithInfo** – for the pattern argument

In addition, the following Xlib functions must support the XLFD conventions:

- **XLoadFont** – for the name argument
- **XListFontsWithInfo** – for the pattern argument
- **XLoadQueryFont** – for the name argument
- **XListFonts** – for the pattern argument

8. BDF Conformance

The bitmap font distribution and interchange format adopted by the X Consortium (BDF V2.1) provides a general mechanism for identifying the font name of an X font and a variable list of font properties, but it does not mandate the syntax or semantics of the font name or the semantics of the font properties that might be provided in a BDF font. This section identifies the requirements for BDF fonts that conform to XLFD.

8.1. XLFD Conformance Requirements

A BDF font conforms to the XLFD specification if and only if the following conditions are satisfied:

- The value for the BDF item **FONT** conforms to the syntax and semantic definition of a XLFD **FontName** string.
- The **FontName** begins with the X **FontNameRegistry** prefix: "−".
- All XLFD **FontName** fields are defined.
- Any FontProperties provided conform in name and semantics to the XLFD **FontProperty** definitions.

A simple method of testing for conformance would entail verifying that the **FontNameRegistry** prefix is the string "−", that the number of field delimiters in the string and coded field values are valid, and that each font property name either matches a standard XLFD property name or follows the definition of a private property.

8.2. FONT_ASCENT, FONT_DESCENT, and DEFAULT_CHAR

FONT_ASCENT, FONT_DESCENT, and DEFAULT_CHAR are provided in the BDF specification as properties that are moved to the **XFontStruct** by the BDF font compiler in generating the X server-specific binary font encoding. If present, these properties shall comply with the following semantic definitions.

8.2.1. FONT_ASCENT

FONT_ASCENT is an integer value (of type INT32) that gives the recommended typographic ascent above the baseline for determining interline spacing. Specific glyphs of the font may extend beyond this. If the current position point for line *n* is at [X,Y], then the origin of the next line $m = n + 1$ (allowing for a possible font change) is [X, Y + FONT_DESCENTn + FONT_ASCENTm].

FONT_ASCENT can be approximated if not provided as a font property, according to the following algorithm:

```
if (FONT_ASCENT undefined) then
  FONT_ASCENT = maximum ascent
```

where maximum ascent is the maximum ascent (above the baseline) in pixels of any glyph in the font.

8.2.2. FONT_DESCENT

FONT_DESCENT is an integer value (of type INT32) that gives the recommended typographic descent below the baseline for determining interline spacing. Specific glyphs of the font may extend beyond this. If the current position point for line *n* is at [X,Y], then the origin of the next line $m = n+1$ (allowing for a possible font change) is [X, Y + FONT_DESCENTn + FONT_ASCENTm].

The logical extent of the font is inclusive between the Y-coordinate values: Y − FONT_ASCENT and Y + FONT_DESCENT + 1.

FONT_DESCENT can be approximated if not provided as a font property, according to the following algorithm:

```
if (FONT_DESCENT undefined) then
  FONT_DESCENT = maximum descent
```

where maximum descent is the maximum descent (below the baseline) in pixels of any glyph in the font.

8.2.3. DEFAULT_CHAR

The DEFAULT_CHAR is an unsigned integer value (of type CARD32) that specifies the index of the default character to be used by the X server when an attempt is made to display an undefined or nonexistent character in the font. (For a font using a 2-byte matrix format, the index bytes are encoded in the integer as byte1 * 65536 + byte2.) If the DEFAULT_CHAR itself specifies an undefined or nonexistent character in the font, then no display is performed.

DEFAULT_CHAR cannot be approximated if not provided as a font property.

Table of Contents