X Logical Font Description Conventions

Version 1.3

MIT X Consortium Standard

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, Version 11, do not 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 fonts available on any given server, in what sizes, styles, etc., so that understandable information can be presented to the user, or that intelligent font fallbacks can be chosen. It is desirable that the most common queries could be accomplished without the overhead of opening each font and inspection of font properties, i.e., through simple **ListFonts** requests. (E.g., 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, character sets available for that family).

This document gives a standard logical font description (XLFD) and conventions to be used in the X 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 via 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, RELATIVE_SETWIDTH, for publishing and other applications to do intelligent font matching or substitution when handling documents created on some foreign server using potentially unknown fonts. In addition, this information is required by certain clients to automatically place subscripts, determine small capital heights, recommended leading, wordspace values, etc.

Examples are for illustrative purposes only.

2. Requirements and Goals

This specification meets the short and long-term goals to have a standard logical font description which:

- provides unique, descriptive font names that support simple pattern matching;
- supports multiple font vendors, arbitrary character sets and encodings;
- is X server and operating/file system independent;
- provides adequate descriptive font information for arbitrarily complex font matching/substitution;
- is extensible.

2.1. Unique, 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/size-dependent font masters, multi-vendor font libraries, etc., 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, 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 a number of 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, while others may desire to perform the mapping dynamically in the server. Provisions must be made in the font name which allows a font request to specify/identify specific charset mappings in server environments where multiple charsets are supported.

2.3. Server, Operating and File System Independent

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).

2.4. Supports Arbitrarily Complex Font Matching/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, so that client applications can derive font-specific formatting/display data, perform font matching/substitution when asked to handle potentially unknown fonts, as required.

2.5. Extensible

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

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 (e.g., **OpenFont**, **ListFonts**, etc.) and a variable list of optional **FontProperties** that describe a font in more detail.

The **FontName** is used in font queries and returned as data in certain X protocol requests. The **FontName** is also specified as the data value for the **FONT** item in the X Consortium Character Bitmap Distribution Format specification (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 **FontProperties** names and associated data values may also appear as items of the **STARTPROPERTIES**...**ENDPROPERTIES** list in the BDF V2.1 specification.

3.1. FontName

The **FontName** is logically composed of two strings: a **FontNameRegistry** prefix, followed by a **FontNameSuffix**. The **FontNameRegistry** is an x-registered-name 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 defined to be the string ‘‘–’’, i.e., ISO 8859-1 HYPHEN (Column/Row 02/13). All **FontNameRegistry** prefixes of the form ‘‘+*version*–’’, where *version* is the version of 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**, in which case numeric property values are represented as signed or unsigned integers as appropriate.

3.1.1. FontName Syntax

The **FontName** is a structured, parsable string (X data 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 (e.g., ‘‘2.0’’)
XFNExtPrefix ::= OCTET – the value of ISO8859-1 PLUS (Column/Row 02/13)
XFNDelim ::= OCTET – the value of ISO8859-1 HYPHEN (Column/Row 02/13)
PrivFontNameRegistry ::= STRING8 – other than those strings reserved by XLFD
PrivFontNameSuffix ::= STRING8

Field values are constructed as strings of ISO8859-1 graphic characters, excluding the following:

- HYPHEN (02/13), the XLFD font name delimiter character;
- QUESTION MARK (03/15) and ASTERISK (02/10), the X protocol fontname wildcard characters.

Alphabetic case distinctions are allowed, but are for human readability concerns only. Conforming X servers will perform matching on font name query/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; e.g., a FAMILY_NAME might be ITC Avant Garde Gothic.

3.1.2. FontName Field Definitions

3.1.2.1. FOUNDRY : x-registered-name

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 (e.g., ITC) but digitized and sold by any number of different type suppliers. Each digital version of the original design will in general be somewhat different in metrics and shape from the idealized original font data, as 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 names, and shall maintain and publish in a timely manner a registry of such registered names for use in XLFD font names and properties.

3.1.2.2. FAMILY_NAME : string

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 (see examples below).

Examples of FAMILY_NAMEs:

Helvetica
ITC Avant Garde Gothic
Times
Times Roman
Bitstream Amerigo
Stone

3.1.2.3. WEIGHT_NAME : string

WEIGHT_NAME is a string that identifies the font’s typographic weight, i.e., the nominal blackness of the font, according to the FOUNDRY’s judgement. This name must be human-understandable and suitable for presentation to a font user.

The interpretation of this field is somewhat problematic, as the typographic judgement of weight has traditionally depended on the overall design of the typeface family in question (i.e., 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 since it is an important part of a font’s complete human-understandable name, but it should not be used for font matching/substitution. 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, for this purpose.

3.1.2.4. SLANT : code-string

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 vertical
‘‘O’’	Oblique	Obliqued upright design, slanted clockwise from vertical
‘‘RI’’	Reverse Italic	Italic design, slanted counter clockwise from vertical
‘‘RO’’	Reverse Oblique	Obliqued upright design, slanted counter clockwise from vertical
‘‘OT’’	Other	Other

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 : string

SETWIDTH_NAME is a string that gives the font's typographic proportionate width, i.e., the nominal width per horizontal unit of the font, according to the FOUNDRY's judgment.

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

Examples of SETWIDTH_NAMEs:

Normal
Condensed
Narrow
Double Wide

3.1.2.6. ADD_STYLE_NAME : string

ADD_STYLE_NAME is a string that identifies additional typographic style information not captured by other fields, but needed to uniquely identify the font.

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

Examples of ADD_STYLE_NAMEs:

Serif
Sans Serif
Informal
Decorated

3.1.2.7. PIXEL_SIZE : integer-string

PIXEL_SIZE is an unsigned integer-string typographic metric in device pixels which gives the body size of the font at a particular POINT_SIZE and RESOLUTION_Y. PIXEL_SIZE normally incorporates additional vertical spacing considered part of the font design. (Note, however, that this value is not necessarily equivalent to the height of the font bounding box). PIXEL_SIZE is in the range zero to a ‘‘very-large-number’’.

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

3.1.2.8. POINT_SIZE : integer-string

POINT_SIZE is an unsigned integer-string typographic metric in device-independent units which gives the body size the font was designed for. This field normally incorporates additional vertical spacing 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 pts = 1 inch) in the range zero to a ‘‘very-large-number’’.

POINT_SIZE and RESOLUTION_Y would be 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 : integer-string

3.1.2.10. RESOLUTION_Y : integer-string

RESOLUTION_X and RESOLUTION_Y are unsigned integer-strings, the horizontal and vertical resolution that the font was designed for, measured in pixels/dots per inch (dpi). Horizontal and vertical values are required since a separate bitmap font must be designed for displays with very different aspect ratios (e.g., 1:1, 4:3, 2:1, etc.).

The separation of pixel/point size and resolution is necessary because X allows for servers with very different video characteristics (e.g., horizontal and vertical resolution, screen and pixel size, pixel shape, etc.) to potentially access the same font library. The font name, then, 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 (about 28 pixels), etc. Further, in servers that implement some or all fonts as continuously scaled 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.

3.1.2.11. SPACING : code-string

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

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 (i.e., all char widths of the font are = max_bounds.width). No other restrictions are placed on the metrics of a monospaced font.
‘‘C’’	CharCell	A monospaced font which follows the standard typewriter character cell model (i.e., the glyphs of the font can be modeled by X clients as ‘‘boxes’’ of the same width and height which 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 (i.e., on a per char basis with positive metrics: 0 <= left-bearing <= right-bearing <= width; with negative metrics: width <= left-bearing <= right-bearing <= 0)–and the vertical extents of the font do not exceed the vertical spacing (i.e., on a per char basis: ascent <= font-ascent and descent <= font-descent). The cell height = font-descent + font-ascent, and width = AVERAGE_WIDTH.

3.1.2.12. AVERAGE_WIDTH : integer-string

AVERAGE_WIDTH is an unsigned integer-string typographic metric value giving the unweighted arithmetic mean width of all glyphs in the font, measured in 1/10th pixels. Note, for monospaced and character cell fonts, this is the width of all glyphs in the font.

3.1.2.13. CHARSET_REGISTRY : x-registered-name

3.1.2.14. CHARSET_ENCODING : registered-name

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.

Although the X protocol does not explicitly have any knowledge about character set encodings, it is expected that server implementers will prefer to embed knowledge of certain proprietary or industry standard charsets into their font library for reasons of performance and convenience. The CHARSET_REGISTRY and CHARSET_ENCODING fields/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/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 # 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 ::= ‘‘.’’--ISO 8859-1 FULL STOP (Column/Row 2/14)
Year ::= STRING8--numeric year (for example, 1989)

The X Consortium shall maintain and publish in a timely manner 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’’.

3.1.3. Examples

The following examples of font names are derived from the screen fonts shipped with the R3 server.

Font	X FontName
75dpi Fonts	
Charter 12pt	-Bitstream-Charter-Medium-R-Normal--12-120-75-75-P-68-ISO8859-1
Charter Bold 12pt	-Bitstream-Charter-Bold-R-Normal--12-120-75-75-P-76-ISO8859-1
Charter BoldItalic 12pt	-Bitstream-Charter-Bold-I-Normal--12-120-75-75-P-75-ISO8859-1C
Charter Italic 12pt	-Bitstream-Charter-Medium-I-Normal--12-120-75-75-P-66-ISO8859-1
Courier 8pt	-Adobe-Courier-Medium-R-Normal--8-80-75-75-M-50-ISO8859-1
Courier 10pt	-Adobe-Courier-Medium-R-Normal--10-100-75-75-M-60-ISO8859-1
Courier 12pt	-Adobe-Courier-Medium-R-Normal--12-120-75-75-M-70-ISO8859-1
Courier 14pt	-Adobe-Courier-Medium-R-Normal--14-140-75-75-M-90-ISO8859-1
Courier 18pt	-Adobe-Courier-Medium-R-Normal--18-180-75-75-M-110-ISO8859-1
Courier 24pt	-Adobe-Courier-Medium-R-Normal--24-240-75-75-M-150-ISO8859-1

Font	**X FontName**
Courier Bold 10pt	-Adobe-Courier-Bold-R-Normal--10-100-75-75-M-60-ISO8859-1
Courier BoldOblique 10pt	-Adobe-Courier-Bold-O-Normal--10-100-75-75-M-60-ISO8859-1
Courier Oblique 10pt	-Adobe-Courier-Medium-O-Normal--10-100-75-75-M-60-ISO8859-1
100dpi Fonts	
Symbol 8pt	-Adobe-Symbol-Medium-R-Normal--11-80-100-100-P-61-Adobe-FONTSPECIFIC
Symbol 10pt	-Adobe-Symbol-Medium-R-Normal--14-100-100-100-P-85-Adobe-FONTSPECIFIC
Symbol 12pt	-Adobe-Symbol-Medium-R-Normal--17-120-100-100-P-95-Adobe-FONTSPECIFIC
Symbol 14pt	-Adobe-Symbol-Medium-R-Normal--20-140-100-100-P-107-Adobe-FONTSPECIFIC
Symbol 18pt	-Adobe-Symbol-Medium-R-Normal--25-180-100-100-P-142-Adobe-FONTSPECIFIC
Symbol 24pt	-Adobe-Symbol-Medium-R-Normal--34-240-100-100-P-191-Adobe-FONTSPECIFIC
Times Bold 10pt	-Adobe-Times-Bold-R-Normal--14-100-100-100-P-76-ISO8859-1
Times BoldItalic 10pt	-Adobe-Times-Bold-I-Normal--14-100-100-100-P-77-ISO8859-1
Times Italic 10pt	-Adobe-Times-Medium-I-Normal--14-100-100-100-P-73-ISO8859-1
Times Roman 10pt	-Adobe-Times-Medium-R-Normal--14-100-100-100-P-74-ISO8859-1

3.2. FontProperties

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/usage 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/display decisions with good typographic results.

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, i.e., private font property names are of the form: ISO8859-1 UNDERSCORE (Column/Row 05/15), followed by the organizational identifier, followed by UNDERSCORE, and terminated with the property name.

The Backus-Naur Form syntax description of X Font Properties is:

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 | COPYRIGHT | AVG_CAPITAL_WIDTH | AVG_LOWERCASE_WIDTH | RELATIVE_SETWIDTH | RELATIVE_WEIGHT | CAP_HEIGHT | SUPERSCRIPT_SIZE | FIGURE_WIDTH | SUBSCRIPT_SIZE | SMALL_CAP_SIZE | NOTICE | DESTINATION
Underscore ::= OCTET-the value of ISO8859-1 UNDERSCORE character (Column/Row 05/15)
OrganizationId ::= STRING8-the registered name of the organization

3.2.1. FOUNDRY : ATOM

As defined in the **FontName**, except the property type is ATOM.

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

3.2.2. FAMILY_NAME : ATOM

As defined in the **FontName**, except the property type is ATOM.

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

3.2.3. WEIGHT_NAME : ATOM

As defined in the **FontName**, except 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 : ATOM

As defined in the **FontName**, except 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 : ATOM

As defined in the **FontName**, except 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 : ATOM

As defined in the **FontName**, except 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 : CARD32

As defined in the **FontName**, except the property type is CARD32.

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)
```

Note that a ‘‘/’’ denotes real division, ‘‘*’’ denotes real multiplication, and ‘‘ROUND’’ denotes a function that rounds its real argument ‘‘a’’ up/down to the next integer, according to x = FLOOR(a + 0.5), where FLOOR is a function that rounds its argument down to an integer.

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

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

3.2.8. POINT_SIZE : CARD32

As defined in the **FontName**, except the property type is CARD32.

X clients requiring device-independent values for EMspace, ENspace and THINspace, can use the following algorithm:

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

Design POINT_SIZE can not be calculated or approximated.

3.2.9. RESOLUTION_X : CARD32

As defined in the **FontName**, except the property type is CARD32.

RESOLUTION_X cannot be calculated or approximated.

3.2.10. RESOLUTION_Y : CARD32

As defined in the **FontName**, except the property type is CARD32.

RESOLUTION_Y cannot be calculated or approximated.

3.2.11. SPACING : ATOM

As defined in the **FontName**, except 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 : CARD32

As defined in the **FontName**, except the property type is CARD32.

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 (all glyph widths in font) * 10)
```

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

X clients requiring 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 : ATOM

As defined in the **FontName**, except the property type is ATOM.

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

3.2.14. CHARSET_ENCODING : ATOM

As defined in the **FontName**, except the property type is ATOM.

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

3.2.15. MIN_SPACE : CARD32

MIN_SPACE is an unsigned integer value that gives the recommended minimum wordspace value to be used with this font.

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

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

3.2.16. NORM_SPACE : CARD32

NORM_SPACE is an unsigned integer value that gives the recommended normal wordspace 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 : CARD32

MAX_SPACE is an unsigned integer value that gives the recommended maximum wordspace 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 : CARD32

END_SPACE is an unsigned integer value 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 : INT32

AVG_CAPITAL_WIDTH is an integer value that gives the unweighted arithmetic mean width of all the capital glyphs in the font, in 1/10th pixels (applies to Latin and non-Latin fonts). For Latin fonts, capitals are the glyphs A-Z. Normally used for font matching/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
  AVG_CAPITAL_WIDTH = ROUND (MEAN (capital glyph widths) * 10)
```

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

3.2.20. AVG_LOWERCASE_WIDTH : INT32

AVG_LOWERCASE_WIDTH is an integer value that gives the unweighted arithmetic mean width of all the lower case glyphs in the font in 1/10th pixels. For Latin fonts, lower case are the glyphs a-z. Normally 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 (lower case exists) then
    AVG_LOWERCASE_WIDTH = ROUND (MEAN (lower case glyph widths) * 10)
  else AVG_LOWERCASE_WIDTH undefined
```

3.2.21. QUAD_WIDTH : INT32 (DEPRECATED)

QUAD_WIDTH was incorrectly defined in the X protocol, and is redundant since all typographic fixed spaces (EM, EN and THIN) are constant for a given font size (i.e., they do not vary according to setwidth). X clients requiring these properties are encouraged to discontinue usage of QUAD_WIDTH and compute these values from other font properties. X clients requiring 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) for this purpose.

See also PIXEL_SIZE, FIGURE_WIDTH, AVERAGE_WIDTH, AVG_CAPITAL_WIDTH and AVG_LOWERCASE_WIDTH font property definitions.

3.2.22. FIGURE_WIDTH : INT32

FIGURE_WIDTH is an integer typographic metric 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 0-9.

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 : INT32

SUPERSCRIPT_X is an integer value 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)
```

Note that TANGENT is a trigonometric function that returns the tangent of its argument (in degrees scaled by 64).

3.2.24. SUPERSCRIPT_Y : INT32

SUPERSCRIPT_Y is an integer value 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 : INT32

SUBSCRIPT_X is an integer value 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 : INT32

SUBSCRIPT_Y is an integer value 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 : CARD32

SUPERSCRIPT_SIZE is an unsigned integer value that gives the recommended body size of synthetic superscripts to be used with this font, in pixels. Note that this will generally be smaller than the size of the current font; i.e., 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 : CARD32

SUBSCRIPT_SIZE is an unsigned integer value 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; i.e., 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 : CARD32

SMALL_CAP_SIZE is an integer value 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 : INT32

UNDERLINE_POSITION is an integer value 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(max_bounds.descent / 2)
```

3.2.31. UNDERLINE_THICKNESS : CARD32

UNDERLINE_POSITION is an unsigned integer value 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 : INT32

STRIKEOUT_ASCENT is an integer value 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 =  max_bounds.ascent
```

3.2.33. STRIKEOUT_DESCENT : INT32

STRIKEOUT_DESCENT is an integer value 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 =  max_bounds.descent
```

3.2.34. ITALIC_ANGLE : INT32

ITALIC_ANGLE is an integer value 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 positon.

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 : CARD32

CAP_HEIGHT is an unsigned integer, the nominal height of the capital letters contained in the font, as specified by the FOUNDRY or typeface designer. Where applicable, it is defined to be the height of the Latin upper case letter X.

CAP_HEIGHT is required by certain clients to compute scale factors and positioning offsets for algorithmically generated glyphs where this information or designed glyphs are not explicitly provided by the font (e.g., small capitals, superiors, inferiors, etc.). Capital 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 capital glyph]
  else CAP_HEIGHT undefined
```

3.2.36. X_HEIGHT : CARD32

X_HEIGHT is a unsigned integer, the nominal height above the baseline of the lower case glyphs contained in the font, as specified by the FOUNDRY or typeface designer. Where applicable, it is defined to be the height of the Latin lower case letter x.

As with Capital height, X_HEIGHT is required by certain clients to compute scale factors for algorithmically generated small capitals, where not explicitly provided by the font resource and 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 (lower case exists) then
    X_HEIGHT = XCharStruct.ascent[some lower case glyph]
  else X_HEIGHT is undefined
```

3.2.37. RELATIVE_SETWIDTH : CARD32

RELATIVE_SETWIDTH is an integer 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 judgement.

The possible values are:

Code	English String	Description
0	undefined	Undefined or unknown
10	UltraCondensed	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	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
```

X clients that wish to obtain a calculated proportionate width of the font (i.e., 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)
```

Note that SETWIDTH is a real with 0 being the ‘‘narrowest’’ calculated setwidth.

3.2.38. RELATIVE_WEIGHT : CARD32

RELATIVE_WEIGHT is an integer 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 judgement.

The possible values are:

Code	English String	Description
0	undefined	Undefined or unknown
10	UltraLight	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
```

3.2.39. WEIGHT : CARD32

Calculated WEIGHT is an unsigned integer, the calculated weight of the font, computed as the ratio of capital stem width to CAP_HEIGHT, in the range 0 to 1000, where zero 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 somewhat subjective. Calculated WEIGHT provides a font-independent way of identifying the weight across all fonts and all font vendors.

3.2.40. RESOLUTION : CARD32 (DEPRECATED)

Independent horizontal and vertical design resolution components are required to accomodate displays with nonsquare aspect ratios and are given by the RESOLUTION_X and RESOLUTION_Y font name fields/properties. The units of the original definition of RESOLUTION are also in conflict with these new properties. X clients are encouraged to discontinue usage of RESOLUTION and to use the appropriate X,Y resolution properties as required.

3.2.41. FACE_NAME : ATOM

FACE_NAME is a human-understandable string that gives the full device-independent typeface name, including the owner, weight, slant, set, etc., but not the resolution, size, etc. Normally used as feedback during font selection.

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

3.2.42. COPYRIGHT : ATOM

COPYRIGHT is a human-understandable string 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 (i.e., it cannot be captured as a comment that could later be ‘‘thrown away’’ for efficiency reasons).

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

3.2.43. NOTICE : ATOM

NOTICE is a human-understandable string 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 can not be calculated or approximated if not provided as a font property.

3.2.44. DESTINATION : CARD32

DESTINATION is an unsigned integer code that gives the font design destination, i.e., 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 (VideoText), etc.

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 String	Description
0	WYSIWYG	A font optimized to match the typographic design and metrics of an equivalent printer font
1	Video Text	A font optimized for screen legibility and readability

3.3. Built-in Font Property Atoms

The following font property atom definitions were predefined in the initial Version 11 of the X protocol:

Font Property	Property Type
MIN_SPACE	CARD32
NORM_SPACE	CARD32

Font Property	**Property Type**
MAX_SPACE	CARD32
END_SPACE	CARD32
SUPERSCRIPT_X	INT32
SUPERSCRIPT_Y	INT32
SUBSCRIPT_X	INT32
SUBSCRIPT_Y	INT32
UNDERLINE_POSITION	INT32
UNDERLINE_THICKNESS	CARD32
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	CARD32
RESOLUTION	CARD32 – deprecated
COPYRIGHT	ATOM
FACE_NAME	ATOM
FAMILY_NAME	ATOM
DEFAULT_CHAR	CARD32

4. Affected Elements of Xlib and the X Protocol

The following X protocol requests must use the font naming conventions:

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

In addition, the following Xlib functions must use the font naming conventions:

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

5. 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 does not mandate the syntax/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.

5.1. XLFD Conformance Requirements

A BDF font conforms to the XLFD V1.3 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 **FontProperties** definitions.

A simple method of testing for conformance would entail first 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.

5.2. 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.

5.2.1. FONT_ASCENT : INT32

FONT_ASCENT is an unsigned integer that gives the recommended typographic ascent above the baseline, for determining interline spacing. Specific glyphs of the font may extent beyond this. If the current position point for line n is at [X,Y], then the origin of the next line $n+1$ (allowing for a possible font change) is [X, Y + FONT_DESCENT $_n$ + FONT_ASCENT $_{n+1}$].

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

```
if (FONT_ASCENT undefined) then
  FONT_ASCENT = max_bounds.ascent
```

5.2.2. FONT_DESCENT : INT32

FONT_DESCENT is an unsigned integer that gives the recommended typographic descent below the baseline, for determining interline spacing. Specific glyphs of the font may extent beyond this. If the current position point for line n is at [X,Y], then the origin of the next line $n+1$ (allowing for a possible font change) is [X, Y + FONT_DESCENT $_n$ + FONT_ASCENT $_{n+1}$].

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 = max_bounds.descent
```

5.2.3. DEFAULT_CHAR : CARD32

DEFAULT_CHAR is an unsigned integer value that gives the default character to be used by the X server when an attempt is made to display an undefined or non-existent character in the font.

The DEFAULT_CHAR is a 16-bit character (not a two byte character). For a font using two byte matrix format, the DEFAULT_CHAR has byte1 in the most significant byte and byte2 in the least significant byte. If the DEFAULT_CHAR itself is undefined or specifies an undefined or nonexistent character in the font, then no display is performed.

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

Table of Contents