The X Font Service Protocol

X Window System Standard

Jim Fulton, Network Computing Devices, Inc.

Version 2.0

Copyright © 1991 Network Computing Devices, Inc.

Copyright © 1994 X Consortium

Permission to use, copy, modify, distribute, and sell this documentation for any purpose is hereby granted without fee, provided that the above copyright notice and this permission notice appear in all copies. Network Computing Devices, Inc. makes no representations about the suitability for any purpose of the information in this document. This documentation is provided "as is" without express or implied warranty.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE X CONSORTIUM BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of the X Consortium shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization from the X Consortium.

Table of Contents

Introduction 2
Architectural Model 3
Font Server Naming 4
    TCP/IP Names 4
    DECnet Names 5
Protocol 5
    Data Types 5
    Requests 14

  Errors 29
  Events 31
Protocol Encoding 32
  Data Types 33
  Requests 37
  Errors 44
  Events 46
Acknowledgements 47
References 47
A. Suggested Licensing Policies 47
B. Implementation Suggestions 48

Introduction

The management of fonts in large, heterogeneous environments is one of the hardest aspects of using the X Window System [1] . Multiple formats and the lack of a consistent mechanism for exporting font data to all displays on a network prevent the transparent use of applications across different display platforms. The X Font Service protocol is designed to address this and other issues, with specific emphasis on the needs of the core X protocol. Upward-compatible changes (typically in the form of new requests) are expected as consensus is reached on new features (particularly outline font support).

Currently, most X displays use network file protocols such as NFS and TFTP to obtain raw font data which they parse directly. Since a common binary format for this data doesn't exist, displays must be able to interpret a variety of formats if they are to be used with different application hosts. This leads to wasted code and data space and a loss of interoperability as displays are used in unforeseen environments.

By moving the interpretation of font data out of the X server into a separate service on the network, these problems can be greatly reduced. In addition, new technologies, such as dynamically generating bitmaps from scaled or outline fonts, can be provided to all displays transparently. For horizontal text, caching techniques and increased processor power can potentially make rasterization more efficient on large, centralized hosts than on individual displays.

Each font server provides sets of fonts that may be listed and queried for header, property, glyph extents, and bitmap information. This data is transmitted over the network using a binary format (with variations to support different bit- and byte-orders) designed to minimize the amount of processing required by the display. Since the font server, rather than the display, is responsible for parsing the raw font data, new formats can be used by all displays by modifying a single font server.

From the user's point of view, font servers are simply a new type of name in the X font path. Network name services allow descriptive names (such as DEPARTMENT-FONTS or APPLICATION-FONTS) to be translated into proper network addresses. X displays send requests to and read replies from the font server rather than reading directly from files. Since the X Font Service protocol is designed to allow subsets of the font data to be requested, displays may easily implement a variety of strategies for fine-grained demand-loading of glyphs.

---

[1] *X Window System* is a trademark of The Open Group.

Architectural Model

In this document, the words "client" and "server" refer to the consumer and provider of a font, respectively, unless otherwise indicated. It is important to note that in this context, the X server is also a font client.

The X Font Service protocol does not require any changes to the core X protocol or to any applications. To the user, font servers are simply additional types of font path elements. As such, X servers may connect to multiple font servers, as shown in Figure 2.1. Although the font protocol is geared towards the X Window System, it may be also used by other consumers of font data (such as printer drivers).

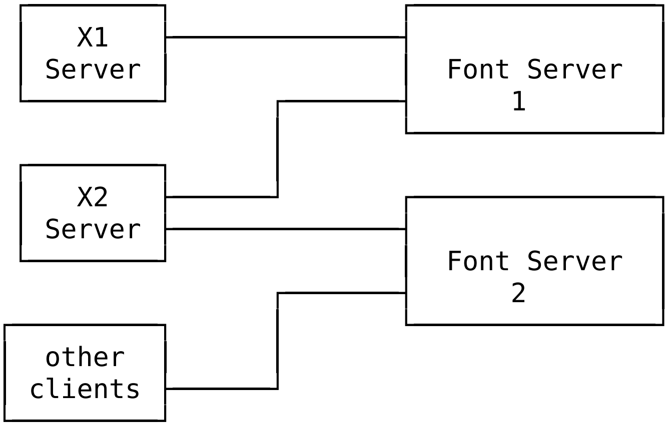
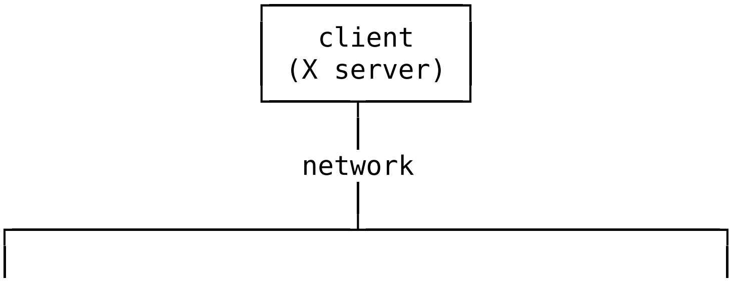


Figure 2.1: Connecting to a Font Server

Clients communicate with the font server using the request/reply/event model over any mutually-understood virtual stream connection (such as TCP/IP, DECnet, [2] etc.). Font servers are responsible for providing data in the bit and byte orders requested by the client. The set of requests and events provided in the first version of the X Font Service protocol is limited to supporting the needs of the bitmap-oriented core X Window System protocol. Extensions are expected as new needs evolve.

A font server reads raw font data from a variety of sources (possibly including other font servers) and converts it into a common format that is transmitted to the client using the protocol described in Section 4. New font formats are handled by adding new converters to a font server, as shown in Figure 2.2.



---

[2] DECnet is a trademark of Digital Equipment Corporation.

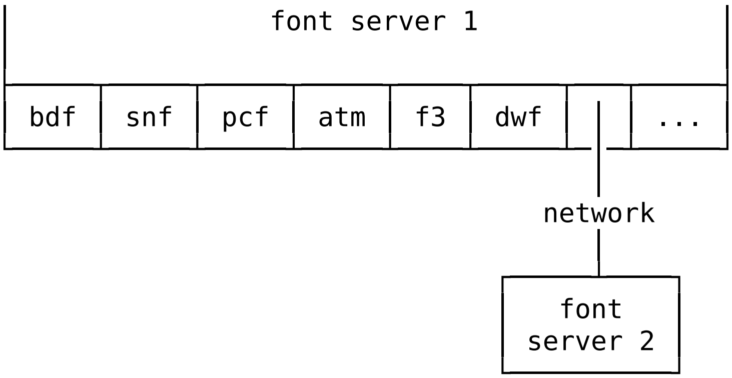


Figure 2.2: Where Font Data Comes From

The server may choose to provide named sets of fonts called "catalogues." Clients may specify which of the sets should be used in listing or opening a font.

An event mechanism similar to that used in the X protocol is provided for asynchronous notification of clients by the server.

Clients may provide authorization data for the server to be used in determining (according to the server's licensing policy) whether or not access should be granted to particular fonts. This is particularly useful for clients whose authorization changes over time (such as an X server that can verify the identity of the user).

Implementations that wish to provide additional requests or events may use the extension mechanism. Adding to the core font service protocol (with the accompanying change in the major or minor version numbers) is reserved to the X Consortium.

Font Server Naming

Font clients that expose font server names to the user are encouraged to provide ways of naming font servers symbolically (e.g. DEPARTMENT-FONTS). However, for environments that lack appropriate name services transport-specific names are necessary. Since these names do occur in the protocol, clients and servers should support at least the applicable formats described below. Formats for additional transports may be registered with the X Consortium.

TCP/IP Names

The following syntax should be used for TCP/IP names:

```
<TCP name   ::=   tcp/  <hostname  :  <ipportnumber  [ /  <cataloguelist ]
```

where <hostname> is either symbolic (such as expo.lcs.mit.edu) or numeric decimal (such as 18.30.0.212). The <ipportnumber> is the port on which the font server is listening for connections. The <cataloguelist> string at the end is optional and specifies a plus-separated list of catalogues that may be requested. For example:

```
tcp/expo.lcs.mit.edu:8012/available+special
tcp/18.30.0.212:7890
```

DECnet Names

The following syntax should be used for DECnet names:

```
    <DECnet name   ::=   decnet/  <nodename   ::font$  <objname
               [ /  <cataloguelist ]
```

where <nodename> is either symbolic (such as SRVNOD) or the numeric decimal form of the DECnet address (such as 44.70). The <objname> is normal, case-insensitive DECnet object name. The <cataloguelist> string at the end is optional and specifies a plus-separated list of catalogues that may be requested. For example:

```
DECNET/SRVNOD::FONT$DEFAULT/AVAILABLE
decnet/44.70::font$other
```

Protocol

The protocol described below uses the request/reply/error model and is specified using the same conventions outlined in Section 2 of the core X Window System protocol [1]:

- Data type names are spelled in upper case with no word separators, as in: FONTID
- Alternate values are capitalized with no word separators, as in: MaxWidth
- Structure element declarations are in lower case with hyphens as word separators, as in: byte-order-msb

Note

Structure element names are referred to in upper case (e.g. BYTE-ORDER-MSB) when used in descriptions to set them off from the surrounding text. When this document is typeset they will be printed in lower case in a distinct font.

- Type declarations have the form "name: type", as in: CARD8: 8-bit byte
- Comma-separated lists of alternate values are enclosed in braces, as in: { Min, MaxWidth, Max }
- Comma-separated lists of structure elements are enclosed in brackets, as in: [byte1: CARD8, byte2: CARD8]

A type with a prefix "LISTof" represents a counted list of elements of that type, as in: LISTofCARD8

Data Types

The following data types are used in the core X Font Server protocol:

```
ACCESSCONTEXT:     ID
```

This value is specified in the CreateAC request as the identifier to be used when referring to a particular AccessContext resource within the server. These resources are used by the server to store client-specified authorization information. This information may be used by the server to determine whether or not the client should be granted access to particular font data.

In order to preserve the integrity of font licensing being performed by the font server, care must be taken by a client to properly represent the identity of the true user of the font. Some font clients will in fact be servers (for example, X servers) requesting fonts for their own clients. Other font clients may be doing work on behalf of a number of different users over time (for example, print spoolers).

`AccessContexts` must be created (with `CreateAC`) and switched among (with `SetAuthorization`) to represent all of these "font users" properly.

```
ALTERNATESERVER:  [ name:  STRING8,
                    subset:  BOOL ]
```

This structure specifies the NAME, encoded in ISO 8859-1 according to Section 3, of another font server that may be useful as a substitute for this font server. The SUBSET field indicates whether or not the alternate server is likely to only contain a subset of the fonts available from this font server. This information is returned during the initial connection setup and may be used by the client to find a backup server in case of failure.

```
AUTH:  [ name:  STRING8,
         data:  LISTofBYTE ]
```

This structure specifies the name of an authorization protocol and initial data for that protocol. It is used in the authorization negotiation in the initial connection setup and in the CreateAC request.

```
BITMAPFORMAT:
```

```
CARD32 containing the following fields defined by the
sets of values given further below
[
  byte-order-msb:      1 bit,
  bit-order-msb:       1 bit,
  image-rect:          2 bits { Min,
                       MaxWidth,
                       Max },
  zero-pad:            4 bits,
  scanline-pad:        2 bits { ScanlinePad8,
                       ScanlinePad16,
                       ScanlinePad32,
                       ScanlinePad64 },
  zero-pad:            2 bits,
```

```
    scanline-unit:       2 bits { ScanlineUnit8,
                         ScanlineUnit16,
                         ScanlineUnit32,
                         ScanlineUnit64 },
    zero-pad:            2 bits,
    zero-pad:            16 bits,
]
```

This structure specifies how glyph images are transmitted in response to `QueryXBitmaps8` and `QueryXBitmaps16` requests.

If the BYTE-ORDER-MSB bit (1 << 0) is set, the Most Significant Byte of each scanline unit is returned first. Otherwise, the Least Significant Byte is returned first.

If the BIT-ORDER-MSB bit (1 << 1) is set, the left-most bit in each glyph scanline unit is stored in the Most Significant Bit of each transmitted scanline unit. Otherwise, the left-most bit is stored in the Least Significant Bit.

The IMAGE-RECT field specifies a rectangle of pixels within the glyph image. It contains one of the following alternate values:

```
ImageRectMin          (0 << 2)
ImageRectMaxWidth     (1 << 2)
ImageRectMax          (2 << 2)
```

For a glyph with extents XCHARINFO in a font with header information XFONTINFO, the IMAGE-RECT values have the following meanings:

`ImageRectMin` - This refers to the minimal bounding rectangle surrounding the inked pixels in the glyph. This is the most compact representation. The edges of the rectangle are:

```
left:     XCHARINFO.LBEARING
right:    XCHARINFO.RBEARING
top:      XCHARINFO.ASCENT
bottom:   XCHARINFO.DESCENT
```

`ImageRectMaxWidth` - This refers to the scanlines between the glyph's ascent and descent, padded on the left to the minimum left-bearing (or 0, whichever is less) and on the right to the maximum right-bearing (or logical-width, whichever is greater). All glyph images share a common horizontal origin. This is a combination of ImageRectMax in the horizontal direction and ImageRectMin in the vertical direction. The edges of the rectangle are:

```
left:         min (XFONTINFO.MIN-BOUNDS.LBEARING, 0)
right:        max (XFONTINFO.MAX-BOUNDS.RBEARING,
                   XFONTINFO.MAX-BOUNDS.WIDTH)
top:               XCHARINFO.ASCENT
bottom:            XCHARINFO.DESCENT
```

ImageRectMax - This refers to all scanlines, from the maximum ascent (or the font ascent, whichever is greater) to the maximum descent (or the font descent, whichever is greater), padded to the same horizontal extents as MaxWidth. All glyph images have the same sized bitmap and share a common origin. This is the least compact representation, but may be the easiest or most efficient (particularly for character cell fonts) for some clients to use. The edges of the rectangle are:

```
left:         min (XFONTINFO.MIN-BOUNDS.LBEARING, 0)
right:        max (XFONTINFO.MAX-BOUNDS.RBEARING,
                   XFONTINFO.MAX-BOUNDS.WIDTH)
top:          max (XFONTINFO.FONT-ASCENT,
                   XFONTINFO.MAX-BOUNDS.ASCENT)
bottom:       max (XFONTINFO.FONT-DESCENT,
                   XFONTINFO.MAX-BOUNDS.DESCENT)
```

The SCANLINE-PAD field specifies the number of bits (8, 16, 32, or 64) to which each glyph scanline is padded before transmitting. It contains one of the following alternate values:

```
ScanlinePad8          (0 << 8)
ScanlinePad16         (1 << 8)
ScanlinePad32         (2 << 8)
ScanlinePad64         (3 << 8)
```

The SCANLINE-UNIT field specifies the number of bits (8, 16, 32, or 64) that should be treated as a unit for swapping. This value must be less than or equal to the number of bits specified by the SCANLINE-PAD. It contains one of the following alternate values:

```
ScanlineUnit8          (0 << 12)
ScanlineUnit16         (1 << 12)
ScanlineUnit32         (2 << 12)
ScanlineUnit64         (3 << 12)
```

BITMAPFORMATs are byte-swapped as CARD32s. All unspecified bits must be zero.

Use of an invalid BITMAPFORMAT causes a Format error to be returned.

BITMAPFORMATMASK: CARD32 mask

This is a mask of bits representing the fields in a BITMAPFORMAT:

```
ByteOrderMask          (1 << 0)
BitOrderMask           (1 << 1)
ImageRectMask          (1 << 2)
ScanlinePadMask        (1 << 3)
ScanlineUnitMask       (1 << 4)
```

Unspecified bits are required to be zero or else a Format error is returned.

BOOL: CARD8

This is a boolean value containing one of the following alternate values:

```
False              0
True               1
```

BYTE: 8-bit value

This is an unsigned byte of data whose encoding is determined by the context in which it is used.

CARD8: 8-bit unsigned integer

CARD16: 16-bit unsigned integer

CARD32: 32-bit unsigned integer

These are unsigned numbers. The latter two are byte-swapped when the server and client have different byte orders.

CHAR2B: [byte1, byte2: CARD8]

This structure specifies an individual character code within either a 2-dimensional matrix (using BYTE1 and BYTE2 as the row and column indices, respectively) or a vector (using BYTE1 and BYTE2 as most- and least-significant bytes, respectively). This data type is treated as a pair of 8-bit values and is never byte-swapped. Therefore, the client should always transmit BYTE1 first.

EVENTMASK: CARD32 mask

This is a mask of bits indicating which of an extension's (or the core's) maskable events the client would like to receive. Each bit indicates one or more events, and a bit value of one indicates interest in a corresponding set of events. The following bits are defined for event masks specified for the core protocol (i.e. an EXTENSION-OPCODE of zero in `SetEventMask` and `GetEventMask` requests):

```
CatalogueListChangeMask         (1 << 0)
FontListChangeMask              (1 << 1)
```

If `CatalogueListChangeMask` is set, client is interested in receiving `CatalogueListNotify` events. If `FontListChangeMask` is set, the client is interested in receiving `FontListNotify` events.

Extensions that provide additional events may define their own event masks. These event masks have their own scope and may use the same bit values as the core or other extensions. All unused bits must be set to zero. In `SetEventMask` requests, if any bits are set that are not defined for the extension (or core) for which this EVENTMASK is intended (according to the EXTENSION- OPCODE given in the `SetEventMask` request), an `EventMask` error is generated. This value is swapped as a CARD32.

FONTID: ID

This is specified by the client in the request `OpenBitmapFont` as the identifier to be used when referring to a particular open font.

ID: CARD32

This is a 32-bit value in which the top 3 bits must be clear, and at least 1 other bit must be set (yielding a range of 1 through 2^29-1). It is specified by the client to represent objects in the server. Identifiers are scoped according to their type are private to the client; thus, the same identifier may be used for both a FONTID and an ACCESSCONTEXT as well as by multiple clients.

An ID of zero is referred to as None.

INT8: 8-bit signed integer

INT16: 16-bit signed integer

INT32: 32-bit signed integer

These are signed numbers. The latter two are byte-swapped when the client and server have different byte orders.

```
OFFSET32:      [  position:     CARD32,
                  length:       CARD32 ]
```

This structure indicates a position and length within a block of data.

```
PROPINFO:     [ offsets:          LISTofPROPOFFSET,
                data:             LISTofBYTE ]
```

This structure describes the list of properties provided by a font. Strings for all of the properties names and values are stored within the data block and are located using a table of offsets and lengths.

This structure is padded to 32-bit alignment.

```
PROPOFFSET:     [ name:          OFFSET32,
                 value:          OFFSET32,
                 type:           CARD8,
                 zero-pad3:      BYTE, BYTE, BYTE ]
```

This structure specifies the position, length, and type of of data for a property.

The NAME field specifies the position and length (which must be greater than zero) of the property name relative to the beginning of the PROPINFO.DATA block for this font. The interpretation of the position and length of the VALUE field is determined by the TYPE field, which contains one of the following alternate values:

```
String          0
```

```
Unsigned        1
Signed          2
```

which have the following meanings:

This property contains a counted string of bytes. The data is stored in the PROPINFO.DATA block beginning at relative byte VALUE.POSITION (beginning with zero), extending for VALUE.LENGTH (at least zero) bytes.

This property contains a unsigned, 32-bit number stored as a CARD32 in VALUE.POSITION (VALUE.LENGTH is zero).

This property contains a signed, 32-bit number stored as an INT32 in VALUE.POSITION (VALUE.LENGTH is zero). This structure is zero-padded to 32-bit alignment.

RANGE: [min-char, max-char: CHAR2B]

This structure specifies a range of character codes. A single character is represented by MIN-CHAR equals MAX-CHAR. If the linear interpretation of MAX-CHAR is less than that of MIN-CHAR, or if MIN-CHAR is less than the font's XFONTINFO.CHAR-RANGE.MIN-CHAR, or if MAX-CHAR is greater than the font's XFONTINFO.CHAR-RANGE.MAX-CHAR, the range is invalid.

```
RESOLUTION:     [ x-resolution:          CARD16,
                  y-resolution:          CARD16,
                  decipoint-size:        CARD16 ]
```

This structure specifies resolution and point size to be used in resolving partially-specified scaled font names. The X-RESOLUTION and Y-RESOLUTION are measured in pixels-per-inch and must be greater than zero. The DECIPOINT-SIZE is the preferred font size, measured in tenths of a point, and must be greater than zero.

STRING8: LISTofCARD8

This is a counted list of 1-byte character codes, typically encoded in ISO 8859-1. A character code "c" is equivalent to a CHAR2B structure whose BYTE1 is zero and whose BYTE2 is "c".

TIMESTAMP: CARD32

This is the number of milliseconds that have passed since a server-dependent origin. It is provided in errors and events and is permitted to wrap.

XCHARINFO: [lbearing, rbearing: INT16, width: INT16, ascent, descent: INT16, attributes: CARD16]

This structure specifies the ink extents and horizontal escapement (also known as the set- or logical width) of an individual character. The first five values represent directed distances in a coordinate system whose origin is aligned with the lower-left edge of the left-most pixel of the glyph baseline (i.e. the baseline falls between two pixels as shown in Figure 3-1 of the "Bitmap Distribution Format 2.1" Consortium standard [2]).

The LBEARING field specifies the directed distance measured to the right from the origin to the left edge of the left-most inked pixel in the glyph.

The RBEARING field specifies the directed distance (measured to the right) from the origin to the right edge of the right-most inked pixel in the glyph.

The WIDTH field specifies the directed distance (measured to the right) from the origin to the position where the next character should appear (called the "escapement point"). This distance includes any whitespace used for intercharacter padding and is also referred to as the "logical width" or "horizontal escapement."

The ASCENT field specifies the directed distance (measured up) from the baseline to the top edge of the top-most inked pixel in the glyph.

The DESCENT field specifies the directed distance (measured down) from the baseline to the bottom edge of the bottom-most inked pixel.

The ATTRIBUTES field specifies glyph-specific information that is passed through the application. If this value is not being used, it should be zero.

The ink bounding box of a glyph is defined to be the smallest rectangle that encloses all of the inked pixels. This box has a width of RBEARING - LBEARING pixels and a height of ASCENT + DESCENT pixels.

```
XFONTINFO:     [ flags:              CARD32,
                 drawing-direction:  { LeftToRight, RightToLeft }
                 char-range:         RANGE,
                 default-char:       CHAR2B,
                 min-bounds:         XCHARINFO,
                 max-bounds:         XCHARINFO,
                 font-ascent:        INT16,
                 font-descent:       INT16,
                 properties:         PROPINFO ]
```

This structure specifies attributes related to the font as a whole.

The FLAGS field is a bit mask containing zero or more of the following boolean values (unspecified bits must be zero):

```
AllCharactersExist     (1 << 0)
InkInside              (1 << 1)
HorizontalOverlap      (1 << 2)
```

which have the following meanings:

AllCharactersExist

> If this bit is set, all of the characters in the range given by CHAR-RANGE have glyphs encoded in the font. If this bit is clear, some of the characters may not have encoded glyphs.

InkInside

> If this bit is set, the inked pixels of each glyph fall within the rectangle described by the font's ascent, descent, origin, and the glyph's escapement point. If this bit is clear, there may be glyphs whose ink extends outside this rectangle.

HorizontalOverlap

> If this bit is set, the two ink bounding boxes (smallest rectangle enclosing the inked pixels) of some pairs of glyphs in the font may overlap when displayed side-by-side (i.e. the second character is imaged at the escapement point of the first) on a common baseline. If this bit is clear, there are no pairs of glyphs whose ink bounding boxes overlap.

The DRAWING-DIRECTION field contains a hint indicating whether most of the character metrics have a positive (or "LeftToRight") logical width or a negative ("RightToLeft") logical width. It contains the following alternate values:

```
LeftToRight          0
RightToLeft          1
```

The CHAR-RANGE.MIN-CHAR and CHAR-RANGE.MAX-CHAR fields specify the first and last character codes that have glyphs encoded in this font. All fonts must have at least one encoded glyph (in which case the MIN-CHAR and MAX-CHAR are equal), but are not required to have glyphs encoded at all positions between the first and last characters.

The DEFAULT-CHAR field specifies the character code of the glyph that the client should substitute for unencoded characters. Requests for extents or bitmaps for an unencoded character generate zero-filled metrics and a zero-length glyph bitmap, respectively.

The MIN-BOUNDS and MAX-BOUNDS fields contain the minimum and maximum values of each of the extents field of all encoded characters in the font (i.e. non-existent characters are ignored).

The FONT-ASCENT and FONT-DESCENT fields specify the font designer's logical height of the font, above and below the baseline, respectively. The sum of the two values is often used as the vertical line spacing of the font. Individual glyphs are permitted to have ascents and descents that are greater than these values.

The PROPERTIES field contains the property data associated with this font.

This structure is padded to 32-bit alignment.

Requests

This section describes the requests that may be sent by the client and the replies or errors that are generated in response. Versions of the protocol with the same major version are required to be upward-compatible.

Every request on a given connection is implicitly assigned a sequence number, starting with 1, that is used in replies, error, and events. Servers are required to generate replies and errors in the order in which the corresponding requests are received. Servers are permitted to add or remove fonts to the list visible to the client between any two requests, but requests must be processed atomically. Each request packet is at least 4 bytes long and contains the following fields:

```
major-opcode:                CARD8
minor-opcode:                CARD8
length:                     CARD16
```

The MAJOR-OPCODE specifies which core request or extension package this packet represents. If the MAJOR-OPCODE corresponds to a core request, the MINOR-OPCODE contains 8 bits of request-specific data. Otherwise, the MINOR-OPCODE specifies which extension request this packet represents. The LENGTH field specifies the number of 4-byte units contained within the packet and must be at least one. If this field contains a value greater than one it is followed by (LENGTH - 1) * 4 bytes of request-specific data. Unless otherwise specified, unused bytes are not required to be zero.

If a request packet contains too little or too much data, the server returns a Length error. If the server runs out of internal resources (such as memory) while processing a request, it returns an Alloc error. If a server is deficient (and therefore non-compliant) and is unable to process a request, it may return an Implementation error. If a client uses an extension request without previously having issued a `QueryExtension` request for that extension, the server responds with a `Request` error. If the server encounters a request with an unknown MAJOR-OPCODE or MINOR-OPCODE, it responds with a `Request` error. At most one error is generated per request. If more than one error condition is encountered in processing a requests, the choice of which error is returned is server-dependent.

Core requests have MAJOR-OPCODE values between 0 and 127, inclusive. Extension requests have MAJOR-OPCODE values between 128 and 255, inclusive, that are assigned by by the server. All MINOR-OPCODE values in extension requests are between 0 and 255, inclusive.

Each reply is at least 8 bytes long and contains the following fields:

```
type:                 CARD8 value of 0
data-or-unused:       CARD8
sequence-number:      CARD16
length:               CARD32
```

The TYPE field has a value of zero. The DATA-OR-UNUSED field may be used to encode one byte of reply-specific data (see Section 5.2 on request encoding). The least-significant 16 bits of the sequence number of the request that generated the reply are stored in the SEQUENCE-NUMBER field. The LENGTH field specifies the number of 4-byte units in this reply packet, including the fields described above, and must be at least two. If LENGTH is greater than two, the fields described above are followed by (LENGTH - 2) * 4 bytes of additional data.

Requests that have replies are described using the following syntax:

```
RequestName
    arg1:  type1
    arg2:  type2
    ...
    argN:  typeN
      =
   result1:  type1
    result2:  type2
    ...
    resultM:  typeM

Errors:  kind1, kind2 ..., kindK

Description
```

If a request does not generate a reply, the"=>" and result lines are omitted. If a request may generate multiple replies, the "=>" is replaced by a "=>+". In the authorization data exchanges in the initial connection setup and the CreateAC request, "->" indicates data sent by the client in response to data sent by the server.

The protocol begins with the establishment of a connection over a mutually-understood virtual stream:

```
open connection
   byte-order:                        BYTE
   client-major-protocol-version:     CARD16
   client-minor-protocol-version:     CARD16
   authorization-protocols:           LISTofAUTH
```

The initial byte of the connection specifies the BYTE-ORDER in which subsequent 16-bit and 32-bit numeric values are to be transmitted. The octal value 102 (ASCII uppercase `B') indicates that the most-significant byte is to be transmitted first; the octal value 154 (ASCII lowercase `l') indicates that the least-significant byte is to be transmitted first. If any other value is encountered the server closes the connection without any response.

> The CLIENT-MAJOR-PROTOCOL-VERSION and CLIENT-MINOR-PROTOCOL-VERSION specify which version of the font service protocol the client would like to use. If the client can support multiple versions, the highest version should be given. This version of the protocol has a major version of 2 and a minor version of 0.
>
> The AUTHORIZATION-PROTOCOLS contains a list of protocol names and optional initial data for which the client can provide information. The server may use this to determine which protocol to use or as part of the initial exchange of authorization data.

```
=
status:                          { Success, Continue,
                                   Busy, Denied }
server-major-protocol-version:  CARD16
server-minor-protocol-version:  CARD16
alternate-servers-hint:         LISTofALTERNATESERVER
authorization-index:            CARD8
authorization-data:             LISTofBYTE
```

The SERVER-MAJOR-PROTOCOL-VERSION and SERVER-MINOR-PROTOCOL-VERSION specify the version of the font service protocol that the server expects from the client. If the server supports the version specified by the client, this version number should be returned. If the client has requested a higher version than is supported by the server, the server's highest version should be returned. Otherwise, if the client has requested a lower version than is supported by the server, the server's lowest version should be returned. It is the client's responsibility to decide whether or not it can match this version of the protocol.

The ALTERNATE-SERVERS-HINT is a list of other font servers that may have related sets of fonts (determined by means outside this protocol, typically by the system administrator). Clients may choose to contact these font servers if the connection is rejected or lost.

The STATUS field indicates whether the server accepted, rejected, or would like more information about the connection. It has one of the following alternate values:

Success	0
Continue	1
Busy	2
Denied	3

If STATUS is Denied, the server has rejected the client's authorization information. If STATUS is Busy, the server has simply decided that it cannot provide fonts to this client at this time (it may be able to at a later time). In both cases, AUTHORIZATION-INDEX is set to zero, no authorization-data is returned, and the server closes the connection after sending the data described so far.

Otherwise the AUTHORIZATION-INDEX is set to the index (beginning with 1) into the AUTHORIZATION-PROTOCOLS list of the protocol that the server will use for this connection. If the server does not want to use any of the given protocols, this value is set to zero. The AUTHORIZATION-DATA field is used to send back authorization protocol-dependent data to the client (such as a challenge, authentication of the server, etc.).

If STATUS is Success, the following section of protocol is omitted. Otherwise, if STATUS is Continue, the server expects more authorization data from the client (i.e. the connection setup is not finished, so no requests or events may be sent):

```
-
more-authorization-data:   STRING8
=
status:                    { Success, Continue,
                           Busy, Denied }
more-authorization-data:   LISTofBYTE
```

The values in STATUS have the same meanings as described above. This section of protocol is repeated until the server either accepts (sets STATUS to Success) or rejects (sets STATUS to Denied or Busy) the connection.

Once the connection has been accepted and STATUS is Success, an implicit AccessContext is created for the authorization data and the protocol continues with the following data sent from the server:

```
=
remaining-length:           CARD32
maximum-request-length:     CARD16
release-number:             CARD32
vendor:                     STRING8
```

The REMAINING-LENGTH specifies the length in 4-byte units of the remaining data to be transmitted to the client. The MAXIMUM-REQUEST-LENGTH specifies the largest request size in 4-byte units that is accepted by the server and must have a value of at least 4096. Requests with a length field larger than this value are ignored and a Length error is returned. The VENDOR string specifies the name of the manufacturer of the font server. The RELEASE-NUMBER specifies the particular release of the server in a manufacturer-dependent manner.

After the connection is established and the setup information has been exchanged, the client may issue any of requests described below:

> `NoOp`
>
> Errors: Alloc
>
> This request does nothing. It is typically used in response to a `KeepAlive` event.

`ListExtensions`

`=>`

> *names*: LISTofSTRING8
>
> Errors: Alloc
>
> This request returns the names of the extension packages that are supported by the server. Extension names are case-sensitive and are encoded in ISO 8859-1.

`QueryExtension`

> *name*: STRING8

`=>`

present: BOOL

major-version: CARD16

minor-version: CARD16

major-opcode: CARD8

first-event: CARD8

number-events: CARD8

first-error: CARD8

number-errors: CARD8

Errors: `Alloc`

This request determines whether or not the extension package specified by NAME (encoded in ISO 8859-1) is supported by the server and that there is sufficient number of major opcode, event, and error codes available. If so, then PRESENT is set to True, MAJOR-VERSION and MINOR-VERSION are set to the respective major and minor version numbers of the protocol that the server would prefer; MAJOR-OPCODE is set to the value to use in extension requests; FIRST-EVENT is set to the value of the first extension-specific event code or zero if the extension does not have any events; NUMBER-EVENTS is set to the number of new events that the event defines; FIRST-ERROR is set to the value of the first extension-specific error code or zero if the extension does not define any new errors; and NUMBER-ERRORS is set to the number of new errors the extension defines.

Otherwise, PRESENT is set to False and the remaining fields are set to zero.

The server is free to return different values to different clients. Therefore, clients must use this request before issuing any of the requests in the named extension package or using the `SetEventMask request to express interest in any of` this extension's events. Otherwise, a `Request` error is returned.

`ListCatalogues`

pattern: STRING8

max-names: CARD32

`=>+`

replies-following-hint: CARD32

names: LISTofSTRING8

Errors: `Alloc`

This request returns a list of at most MAX-NAMES names of collections (called catalogues) of fonts that match the specified PATTERN. In the pattern (which is encoded in ISO 8859-1), the `?' character (octal 77) matches any single character; the `*' character (octal 52) matches any series of zero or more characters; and alphabetic characters match either upper- or lowercase. The returned NAMES are encoded in ISO 8859-1 and may contain mixed character cases.

If PATTERN is of zero length or MAX-NAMES is equal to zero, one reply containing a zero-length list of names is returned. This may be used to synchronize the client with the server.

Servers are free to add or remove catalogues to the set returned by `ListCatalogues` between any two requests. This request is not cumulative; repeated uses are processed in isolation and do result in an iteration through the list.

To reduce the amount of buffering needed by the server, the list of names may be split across several reply packets, so long as the names arrive in the same order that they would have appeared had they been in a single packet. The REPLIES-FOLLOWING-HINT field in all but the last reply contains a positive value that specifies the number of replies that are likely, but not required, to follow. In the last reply, which may contain zero or more names, this field is set to zero.

`SetCatalogues`

names: LISTofSTRING8

Errors: `Alloc` , `Name`

This request sets the list of catalogues whose fonts should be visible to the client. The union of the fonts provided by each of the named catalogues forms the set of fonts whose names match patterns in `ListFonts` , `ListFontsWithXInfo` , and `OpenBitmapFont` requests. The catalogue names are case-insensitive and are encoded in ISO 8859-1. A zero-length list resets the client's catalogue list to the server-dependent default.

If any of the catalogue names are invalid, a `Name` error is returned and the request is ignored.

`GetCatalogues`

`=>`

names: LISTofSTRING8

Errors: `Alloc`

This request returns the current list of catalogue names (encoded in ISO 8859-1) associated with the client. These catalogues determine the set of fonts that are visible to `ListFonts`, `ListFontsWithXInfo`, and `OpenBitmapFont`. A zero-length list indicates the server's default set of fonts. Catalogue names are case-insensitive and may be returned in mixed case.

`SetEventMask`

extension-opcode: CARD8

event-mask: EVENTMASK

Errors: `EventMask` , `Request`

This request specifies the set of maskable events that the extension indicated by EXTENSION-OPCODE (or zero for the core) should generate for the client. Event masks are limited in scope to the extension (or core) for which they are defined, so expressing interest in events from one or more extensions requires multiple uses of this request.

The default event mask if `SetEventMask` has not been called is zero, indicating no interest in any maskable events. Some events are not maskable and cannot be blocked.

If EXTENSION-OPCODE is not a valid extension opcode previously returned by `QueryExtension` or zero, a `Request` error is returned. If EVENT-MASK contains any bits that do not correspond to valid events for the specified extension (or core), an `EventMask` error is returned and the request is ignored.

`GetEventMask`

extension-opcode: CARD8

`=>`

event-mask: EVENTMASK

Errors: `Request`

This request returns the set of maskable core events the extension indicated by EXTENSION-OPCODE (or the core if zero) should generate for the client. Non-maskable events are always sent to the client.

If EXTENSION-OPCODE is not a valid extension opcode previously returned by `QueryExtension` or zero, a `Request` error is returned.

`CreateAC`

ac: ACCESSCONTEXT

authorization-protocols: LISTofAUTH

`=>`

status: { Success, Continue, Denied }

authorization-index: CARD8

authorization-data: LISTofBYTE

Errors: `IDChoice`

This request creates a new `AccessContext` object within the server containing the specified authorization data. When this `AccessContext` is selected by the client using the `SetAuthorization` request, the data may be used by the server to determine whether or not the client should be granted access to particular font information.

If STATUS is Denied, the server rejects the client's authorization information and does not associate AC with any valid `AccessContext` . In this case, AUTHORIZATION-INDEX is set to zero, and zero bytes of AUTHORIZATION-DATA is returned.

Otherwise, AUTHORIZATION-INDEX is set to the index (beginning with 1) into the AUTHORIZATION-PROTOCOLS list of the protocol that the server will use for this connection. If the server does not want to use any of the given protocols, this value is set to zero. The AUTHORIZATION-DATA field is used to send back authorization protocol-dependent data to the client (such as a challenge, authentication of the server, etc.).

If STATUS is Continue, the client is expected to continue the request by sending the following protocol and receiving the indicated response from the server. This continues until STATUS is set to either Success or Denied.

```
-
more-authorization-data:          STRING8
=
status:                           { Success, Continue, Denied }
more-authorization-data:          LISTofBYTE
```

Once the connection has been accepted and STATUS is Success, the request is complete.

If AC is not in the range [1..2^29-1] or is already associated with an access context, an IDChoice error is returned.

`FreeAC`

ac: ACCESSCONTEXT

Errors: `AccessContext` , `Alloc`

This request indicates that the specified AC should no longer be associated with a valid access context. If AC is also the current `AccessContext` (as set by the `SetAuthorization` request), an implicit `SetAuthorization` of None is done to restore the `AccessContext` established for the initial connection setup. Operations on fonts that were opened under AC are not affected. The client may reuse the value of AC in a subsequent `CreateAC` request.

If AC isn't associated with any valid authorization previously created by `CreateAC` , an `AccessContext` error is returned.

`SetAuthorization`

ac: ACCESSCONTEXT

Errors: `AccessContext`

This request sets the `AccessContext` to be used for subsequent requests (except for `QueryXInfo`, `QueryXExtents8`, `QueryXExtents16`, `QueryXBitmaps8`, `QueryXBitmaps16` and `CloseFont` which are done under the `AccessContext` of the corresponding `OpenBitmapFont` ")." An AC of None restores the `AccessContext` established for the initial connection setup.

If AC is neither None nor a value associated with a valid `AccessContext` previously created by `CreateAC`, an `AccessContext` error is returned.

`SetResolution`

resolutions: LISTofRESOLUTION

Errors: `Resolution`, `Alloc`

This request provides a hint as to the resolution and preferred point size of the drawing surfaces for which the client will be requesting fonts. The server may use this information to set the RESOLUTION_X and RESOLUTION_Y fields of scalable XLFD font names, to order sets of names based on their resolutions, and to choose the server-dependent instance that is used when a partially-specified scalable fontname is opened.

If a zero-length list of RESOLUTIONS is given, the server-dependent default value is restored. Otherwise, if elements of all of the specified RESOLUTIONS are non-zero, the default resolutions for this client are changed.

If a RESOLUTION entry contains a zero, a Resolution error is returned and the default resolutions are not changed.

`GetResolution`

`=>`

resolutions: LISTofRESOLUTION

Errors: `Alloc`

This request returns the current list of default resolutions. If a client has not performed a `SetResolution`, a server-dependent default value is returned.

`ListFonts`

pattern: STRING8

max-names: CARD32

`=>+`

replies-following-hint: CARD32

names: LISTofSTRING8

Errors: `Alloc`

This request returns a list of at most MAX-NAMES font names that match the specified PATTERN, according to matching rules of the X Logical Font Description Conventions [3]. In the pattern (which is encoded in ISO 8859-1) the `?' character (octal 77) matches any single character; the `*' character (octal 52) matches any series of zero or more characters; and alphabetic characters match either upper- or lowercase. The returned NAMES are encoded in ISO 8859-1 and may contain mixed character cases. Font names are not required to be in XLFD format.

If PATTERN is of zero length or MAX-NAMES is equal to zero, one reply containing a zero-length list of names is returned. This may be used to synchronize the client with the server.

Servers are free to add or remove fonts to the set returned by `ListFonts` between any two requests. This request is not cumulative; repeated uses are processed in isolation and do result in an iteration through the list.

To reduce the amount of buffering needed by the server, the list of names may be split across several reply packets, so long as the names arrive in the same order that they would have appeared had they been in a single packet. The REPLIES-FOLLOWING-HINT field in all but the last reply contains a positive value that specifies the number of replies that are likely, but not required, to follow. In the last reply, which may contain zero or more names, this field is set to zero.

`ListFontsWithXInfo`

pattern: STRING8

pattern: STRING8

pattern: STRING8

max-names: CARD32

`=>+`

replies-following-hint: CARD32

info: XFONTINFO

name: STRING8

Errors: `Alloc`

This request is similar to `ListFonts` except that a separate reply containing the name, header, and property data is generated for each matching font name. Following these replies, if any, a final reply containing a zero-length NAME and no INFO is sent.

The REPLIES-FOLLOWING-HINT field in all but the last reply contains a positive value that specifies the number of replies that are likely, but not required, to follow. In the last reply, this field is set to zero.

If PATTERN is of zero length or if MAX-NAMES is equal to zero, only the final reply containing a zero-length NAME and no INFO is returned. This may be used to synchronize the client with the server.

`OpenBitmapFont`

fontid: FONTID

pattern: STRING8

format-mask: BITMAPFORMATMASK

format-hint: BITMAPFORMAT

`=>`

otherid: FONTID or None

otherid-valid: BOOL

cachable: BOOL

Errors: `IDChoice`, `Name`, `Format`, `AccessContext`, `Alloc`

This request looks for a server-dependent choice of the font names that match the specified PATTERN according to the rules described for `ListFonts .` If no matches are found, a `Name` error is returned. Otherwise, the server attempts to open the font associated with the chosen name.

Permission to access the font is determined by the server according the licensing policy used for this font. The server may use the client's current `AccessContext` (as set by the most recent `SetAuthorization` request or the original connection setup) to determine any client-specific sets of permissions. After the font has been opened, the client is allowed to specify a new `AccessContext` with `SetAuthorization` or release the `AccessContext` using `FreeAC` . Subsequent `QueryXInfo`, `QueryXExtents8`, `QueryXExtents16`, `QueryXBitmaps8`, `QueryXBitmaps16` and `CloseFont` requests on this FONTID are performed according to permissions granted at the time of the `OpenBitmapFont` request.

If the server is willing and able to detect that the client has already opened the font successfully (possibly under a different name), the OTHERID field may be set to one of the identifiers previously used to open the font. The OTHERID-VALID field indicates whether or not OTHERID is still associated with an open font: if it is True, the client may use OTHERID as an alternative to FONTID. Otherwise, if OTHERID-VALID is False, OTHERID is no longer open but has not been reused by a subsequent `OpenBitmapFont` request.

If OTHERID is set to None, then OTHERID-VALID should be set to False.

The FORMAT-MASK indicates which fields in FORMAT-HINT the client is likely to use in subsequent `GetXBitmaps8` and `GetXBitmaps16` requests. Servers may wish to use this information to precompute certain values.

If CACHABLE is set to True, the client may cache the font (so that redundant opens of the same font may be avoided) and use it with all `AccessContexts` during the life of the client without violating the font's licensing policy. This flag is typically set whenever a font is unlicensed or is licensed on a per-display basis. If CACHABLE is False, the client should reopen the font for each `AccessContext` .

The server is permitted to add to or remove from the set of fonts returned by `ListFonts` between any two requests, though mechanisms outside the protocol. Therefore, it is possible for this request (which is atomic) to return a different font than would result from separate a `ListFonts` followed by an `OpenBitmapFont` with a non-wildcarded font name.

If FONTID is not in the range [1..2^29-1] or if it is already associated with an open font, an `IDChoice` error is returned. If no font is available that matches the specified PATTERN, a `Name` error is returned. If the font is present but the client is not permitted access, an `AccessContext` error is returned. If FORMAT-MASK has any unspecified bits set or if any of the fields in FORMAT-HINT indicated by FORMAT-MASK are invalid, a `Format` error is returned.

`QueryXInfo`

fontid: FONTID

=>

info: XFONTINFO

Errors: `Font`, `Alloc`

This request returns the font header and property information for the open font associated with FONTID.

If FONTID is not associated with any open fonts, a `Font` error is returned.

`QueryXExtents8`

fontid: FONTID

range: BOOL

chars: STRING8

=>

extents: LISTofXCHARINFO

Errors: `Font`, `Range`, `Alloc`

This request is equivalent to `QueryXExtents16` except that it uses 1-byte character codes.

`QueryXExtents16`

fontid: FONTID

range: BOOL

chars: LISTofCHAR2B

`=>`

extents: LISTofXCHARINFO

Errors: `Font`, `Range`, `Alloc`

This request returns a list of glyph extents from the open font associated with FONTID for the series of characters specified by RANGE and CHARS.

If RANGE is True, each succeeding pair of elements in CHARS is treated as a range of characters for which extents should be returned. If CHARS contains an odd number of elements, the font's XFONTINFO.CHAR-RANGE.MAX-CHAR is implicitly appended to the list. If CHARS contains no elements, the list is implicitly replaced with the font's XFONTINFO.CHAR-RANGE. If any of the resulting character ranges are invalid, a Range error is returned. Otherwise, the character ranges are concatenated in the order given by CHARS to produce a set of character codes for which extents are returned.

If RANGE is False, then CHARS specifies the set of character codes for which extents are returned. If CHARS is of zero length, then a zero-length list of extents is returned.

The extents for each character code in the resulting set (which may contain duplicates) are returned in the order in which the character codes appear in the set. At least one metric for each character shall be non-zero unless the character is not encoded in the font, in which case all-zero metrics are returned. A blank, zero-width character can be encoded with non-zero but equal left and right bearings.

If FONTID is not associated with any open fonts, a `Font` error is returned. If RANGE is True and CHARS contains any invalid ranges, a `Range` error is returned.

`QueryXBitmaps8`

fontid: FONTID

range: BOOL

chars: STRING8

format: BITMAPFORMAT

`=>+`

replies-following-hint: CARD32

offsets: LISTofOFFSET32

bitmaps: LISTofBYTE

Errors: `Font`, `Range`, `Format`, `Alloc`

This request is equivalent to `QueryXBitmaps16` except that it uses 1-byte character codes.

`QueryXBitmaps16`

fontid: FONTID

range: BOOL

chars: LISTofCHAR2B

format: BITMAPFORMAT

`=>+`

replies-following-hint: CARD32

offsets: LISTofOFFSET32

bitmaps: LISTofBYTE

Errors: `Font`, `Range`, `Format`, `Alloc`

This request returns a list of glyph bitmaps from the open font associated with FONTID for the series of characters specified by RANGE and CHARS.

If RANGE is True, each succeeding pair of elements in CHARS is treated as a range of characters for which bitmaps should be returned. If CHARS contains an odd number of elements, the font's XFONTINFO.CHAR-RANGE.MAX-CHAR is implicitly appended to the list. If CHARS contains no elements, the list is implicitly replaced with the font's XFONTINFO.CHAR-RANGE. If any of the resulting character ranges are invalid, a Range error is returned. Otherwise, the character ranges are concatenated in the order given by CHARS to produce a set of character codes for which bitmaps are returned.

If RANGE is False, then CHARS specifies the set of character codes for which bitmaps are returned. If CHARS is of zero length, then a single reply containing a zero-length list of offsets and bitmaps is returned.

If any of the resulting character ranges are invalid, a Range error is returned. Otherwise, the resulting character ranges are concatenated in the order given by CHARS to produce a set of character codes for which bitmaps are returned.

The server is free to return the glyph bitmaps in multiple replies to reduce the amount of buffering that is necessary. In this situation, the set of characters obtained above is partitioned into an implementation-dependent number of ordered, non-overlapping subsets containing runs of one or more consecutive characters. The global ordering of characters must be maintained such that concatenating the subsets in order that they were produced yields the original set. A reply is generated for each subset, in the order that it was produced.

For each character in a subset, an image of that character's glyph is described by a rectangle of bits corresponding to the pixels specified by FORMAT.IMAGE-RECT. Within the image, set and clear bits represent inked and non-inked pixels, respectively.

Each scanline of a glyph image, from top to bottom, is zero-padded on the right to a multiple of the number of bits specified by FORMAT.SCANLINE-PAD. The scanline is then divided from left to right into a sequence of FORMAT.SCANLINE-UNIT bits. The bits of each unit are then arranged such that the left-most pixel is stored in the most- or least-significant bit, according to FORMAT.BIT-ORDER-MSB. The bytes of each unit are then arranged such that the most- or least-significant byte, according to FORMAT.BYTE-ORDER-MSB, is transmitted first. Finally, the units are arranged such that the left-most is transmitted first and the right-most is transmitted last.

The individual images within a subset are then concatenated in a server-dependent order to form the BITMAPS data of the reply. If a glyph image is duplicated within a reply, the server is free to return fewer (but at least one) copies of the image. If a character is not encoded within the font, a zero-length bitmap is substituted for this character. Each glyph image must begin at a bit position that is a multiple of the FORMAT.SCANLINE-UNIT.

The OFFSETS array in a reply contains one entry for each character in the subset being returned, in the order that the characters appear in the subset. Each entry specifies the starting location in bytes and size in bytes of the corresponding glyph image in the BITMAPS data of that reply (i.e. an offset may not refer to data in another reply).

The REPLIES-FOLLOWING-HINT field in all but the last reply contains a positive value that specifies the number of replies that are likely, but not required, to follow. In the last reply, which may contain data for zero or more characters, this field is set to zero.

If FONTID is not associated with any open fonts, a Font error is returned. If RANGE is True and CHARS contains any invalid ranges, a Range error is returned. If FORMAT is invalid, a Format error is returned.

`CloseFont`

fontid: FONTID

Errors: `Font`, `Alloc`

This request indicates that the specified FONTID should no longer be associated with an open font. The server is free to release any client-specific storage or licenses allocated for the font. The client may reuse the value of FONTID in a subsequent `OpenBitmapFont` request.

If FONTID is not associated with any open fonts, a `Font` error is returned.

`close connection`

When a connection is closed, a `CloseFont` is done on all fonts that are open on the connection. In addition, the server is free to release any storage or licenses allocated on behalf of the client that made the connection.

Errors

All errors are at least 16 bytes long and contain the following fields:

type: CARD8 value of 1

error-code: CARD8

sequence-number: CARD16

length: CARD32

timestamp: TIMESTAMP

major-opcode: CARD8

minor-opcode: CARD8

data-or-unused: CARD16

The TYPE field has a value of one. The ERROR-CODE field specifies which error occurred. Core errors codes are in the range 0 through 127, extension error codes are in the range 128 through 255. The SEQUENCE-NUMBER field contains the least significant 16 bits of the sequence number of the request that caused the error. The LENGTH field specifies the length of the error packet in 4-byte units and must have a value of at least 4. The TIMESTAMP specifies the server time when the error occurred. The MAJOR-OPCODE and MINOR-OPCODE (zero for core requests) fields specify the type of request that generated the error. The DATA-OR-UNUSED field may be used for 16 bits of error-specific information. If LENGTH is greater than four, these fields are followed by (LENGTH - 4) * 4 bytes of extra data.

The following errors are defined for the core protocol:

`Request`

data-or-unused: CARD16 unused

This error is generated by any request that has an unknown combination of major and minor request numbers, or by any extension request that is issued before a `QueryExtension` of that extension.

`Format`

data-or-unused: CARD16 unused

format: BITMAPFORMAT bad format value

This error is generated by the use of an invalid BITMAPFORMAT in the `OpenBitmapFont`, `QueryXBitmaps8`, and `QueryXBitmaps16` requests. The value that caused the error is included as extra data.

`Font`

data-or-unused: CARD16 unused

fontid: FONTID bad font identifier

This error is generated by an invalid FONTID in the `QueryXInfo`, `QueryXExtents8`, `QueryXExtents16`, `QueryXBitmaps8`, `QueryXBitmaps16` and `CloseFont` requests. The value that caused the error is included as extra data.

`Range`

data-or-unused: CARD16 unused

range: RANGE bad range

This error is generated by an invalid RANGE in the `QueryXExtents8`, `QueryXExtents16`, `QueryXBitmaps8` and `QueryXBitmaps16` requests. The value that caused the error is included as extra data.

`EventMask`

data-or-unused: CARD16 unused

event-mask: EVENTMASK bad event mask

This error is generated by an invalid EVENTMASK in the `SetEventMask` request. The value that caused the error is included as extra data.

`AccessContext`

data-or-unused: CARD16 unused

ac: ACCESSCONTEXT unaccepted `AccessContext`

This error is generated by an invalid ACCESSCONTEXT in the `FreeAC` or `SetAuthorization` request or by an `OpenBitmapFont` request performed without sufficient authorization. In the first two cases, the ACCESSCONTEXT of the errant request is returned as extra data. In the third case, the current ACCESSCONTEXT is returned as extra data.

`IDChoice`

data-or-unused: CARD16 unused

id: ID bad identifier

This error is generated by an invalid or already associated ACCESSCONTEXT identifier in a `CreateAC` request or FONTID identifier in an `OpenBitmapFont` request. The value that caused the error is included as extra data.

`Name`

data-or-unused: CARD16 unused

This error is generated by a font name pattern that matches no fonts in an `OpenBitmapFont` request or no catalogue names in a `SetCatalogues` request.

`Resolution`

data-or-unused: CARD16 X value of errant resolution

y-resolution: CARD16 Y value of errant resolution *point-size*: CARD16 point size of errant resolution

This error is generated in response to an invalid RESOLUTION structure in a `SetResolution` request. The value that caused the error is included in the DATA-OR-UNUSED field and as extra data.

`Alloc`

data-or-unused: CARD16 unused

This error is generated by any request for which the server lacks sufficient resources (especially memory).

`Length`

data-or-unused: CARD16 unused

length: CARD32 bad length value

This error is generated by any request that has a length field greater than (MAXIMUM-REQUEST-LENGTH * 4) bytes. The value that caused the error is included as extra data.

`Implementation`

data-or-unused: CARD16 unused

This error may be generated in response to any request that the server is unable to process because it is deficient. Use of this error is highly discouraged and indicates lack of conformance to the protocol. Additional errors may be defined by extensions.

Events

Events may be generated in response to requests or at the server's discretion after the initial connection setup information has been exchanged. Each event is at least 12 bytes long and contains the following fields:

type: CARD8 value of 2

event-code: CARD8

sequence-number: CARD16

length: CARD32

timestamp: TIMESTAMP

The TYPE field contains the value 2. The EVENT-CODE field specifies the number of the event and is in the range 0-127 for core events or the range 128-255 for extensions. The SEQUENCE-NUMBER field specifies the least significant 16 bits of the sequence number of the last request to have been processed by the server. The LENGTH field specifies the number of 4-byte units in this event packet and must always have a value of at least 3. The TIMESTAMP field specifies the server time when the event occurred. If LENGTH is greater than three, these fields are followed by (LENGTH - 3) * 4 bytes of additional data.

Events are described using the following syntax:

```
EventName
        arg1: type1
        ...
        argN: typeN

         Description
```

If an event does not provide any extra arguments, the *arg1*...*argN* lines are omitted from the description.

The core X Font Service protocol defines the following events:

`KeepAlive`

This unsolicited, nonmaskable event may be sent by the server to verify that the connection has not been broken (for transports that do not provide this information). Clients should acknowledge receipt of this request by sending any request (such as `NoOp` ")."

`CatalogueListNotify`

added: BOOL

deleted: BOOL

This event is sent to clients that have included `CatalogueListChangeMask` in their core event mask whenever the list of catalogues that are available has changed. The ADDED field is True if new catalogues have been added to the server, otherwise it is False. The DELETED field is True if any existing catalogues have been removed from the server, otherwise it is False.

`FontListNotify`

added: BOOL

deleted: BOOL

This event is sent to clients that have included `FontListChangeMask` in their event mask whenever the list of fonts that are provided by the currently selected catalogues has changed. The ADDED field is True if new fonts have been added to any of the catalogues currently used by the client, otherwise it is False. The DELETED field is True if any existing fonts have been removed from any of catalogues used by the client, otherwise it is False.

Additional events may be defined by extensions.

Protocol Encoding

Numbers that are prefixed with "#x" are in hexadecimal (base 16). All other numbers are in decimal. Requests, replies, errors, events, and compound types are described using the syntax:

```
    Name
     count          contents     name
     ...
     count          contents     name
```

where COUNT is the number of bytes in the data stream occupied by this field, CONTENTS is the name of the type as given in Section 4 or the value if this field contains a constant, and NAME is a description of this field.

Objects containing counted lists use a lowercase single-letter variable (whose scope is limited to the request, reply, event, or error in which it is found) to represent the number of objects in the list. These variables, and any expressions in which they are used, should be treated as unsigned integers. Multiple copies of an object are indicated by CONTENTS prefix "LISTof".

Unused bytes (whose value is undefined) will have a blank CONTENTS field and a NAME field of "unused". Zeroed bytes (whose value must be zero) will have a blank CONTENTS field and a NAME field of "zero". The expression pad(e) refers to the number of bytes needed to round a value "e" up to the closed multiple of four:

```
     pad(e) = (4 - (e mod 4)) mod 4
```

Data Types

```
ACCESSCONTEXT

4 CARD32 access context with at least one of the following bits set:

#x1fffffff

but none of the following bits set:

#xe0000000              zero


ALTERNATESERVER
1     BOOL              subset
1     n                 length of name
n     STRING8           name
p                       unused, p=pad(n+2)

AUTH

2     n                 length of name
2     d                 length of data
n     STRING8           name
p                       unused, p=pad(n)
d     STRING8           data
q                       unused, q=pad(d)
```

```
BITMAPFORMAT

4 CARD32 value, union of the following bits:
     #x00000001           ByteOrderMSB
     #x00000002           BitOrderMSB
     #x00000000           ImageRectMin
     #x00000004           ImageRectMaxWidth
     #x00000008           ImageRectMax
     #x00000000           ScanlinePad8
     #x00000100           ScanlinePad16
     #x00000200           ScanlinePad32
     #x00000300           ScanlinePad64
     #x00000000           ScanlineUnit8
     #x00001000           ScanlineUnit16
     #x00002000           ScanlineUnit32
     #x00003000           ScanlineUnit64

except for the following bits which must be zero:

     #xffffccf0           zero

and the following of which at most one bit may be set:

     #x0000000c at most one bit can be set


BITMAPFORMATMASK

4 CARD32 value, mask of the following bits:

     #x00000001           ByteOrderMask
     #x00000002           BitOrderMask
     #x00000004           ImageRectMask
     #x00000008           ScanlinePadMask
     #x00000010           ScanlineUnitMask

except for the following bits which must be zero:

     #xffffffe0           zero

BOOL

1    BOOL       boolean, one of the following values:
0    False
1    True

BYTE
1    BYTE       unsigned byte of data

CARD8
1    CARD8       8-bit unsigned integer

CARD16
2    CARD16      16-bit unsigned integer
```

```
CARD32
4    CARD32      32-bit unsigned integer

CHAR2B
1    CARD8       byte1
1    CARD8       byte2

EVENTMASK
4    CARD32 event mask
     for core events, this is union of the following bits:

     #00000001   CatalogueListChangeMask
     #00000002   FontListChangeMask

but none of the following bits set:

     #fffffffc

extensions define their own sets of bits

FONTID

4 CARD32 font identifier with at least one of
the following bits set:

     #x1fffffff

but none of the following bits set:

     #xe0000000   zero

INT8
1    INT8         8-bit signed integer

INT16
2    INT16         16-bit signed integer

INT32
4    INT32         32-bit signed integer

OFFSET32
4    CARD32         position (or integer value)
4    CARD32         length

PROPINFO
4    n                 number of PROPOFFSET components
4    m                 number of bytes of property data
20*n PROPOFFSET         property offsets into data block
m    LISTofBYTE         property data block

PROPOFFSET
8    OFFSET32         name in data block
8    OFFSET32         value in data block
```

```
1 CARD8 type, one of the following values:
0    String
1    Unsigned
2    Signed
3    zero

RANGE
2    CHAR2B                 minimum character code
2    CHAR2B         maximum character code

RESOLUTION
2    CARD16 x resolution in pixels per inch
2    CARD16 y resolution in pixels per inch
2    CARD16 point size in decipoints

STRNAME
1    n length of name
n    STRING8 name

STRING8
n    LISTofBYTE         array of 8-bit character values

TIMESTAMP
4    CARD32         milliseconds since server time origin

XCHARINFO
2    INT16         left bearing
2    INT16         right bearing
2    INT16         width
2    INT16         ascent
2    INT16         descent
2    CARD16         attributes

XFONTINFO
4 CARD32 flags, union of the following bits:
     #x00000001         AllCharactersExist
     #x00000002         InkInside
     #x00000004         HorizontalOverlap

but none of the following bits set:

     #xfffffff8         zero
4    RANGE              range of characters in font
1    CARD8              drawing direction
     0             LeftToRight
     1             RightToLeft
1                       unused
2    CHAR2B             default character
12   XCHARINFO          minimum bounds
12   XCHARINFO          maximum bounds
2    INT16              font ascent
2    INT16              font descent
n    PROPINFO           property data
```

Requests

open connection

```
1     BYTE                    byteorder, one of the values:
      #x42                    MostSignificant Byte first
      #x6c                    LeastSignificant Byte first
1     CARD8                   numberof auth in auth-data
2     2                       client-major-protocol-version
2     0                       client-minor-protocol-version
2     a/4 lengthof            auth-data
a     LISTofAUTH              auth-data
=
2     CARD16                  status
0                             Success
1                             Continue
2                             Busy
3                             Denied
2     2                       major version
2     0                       version
1     CARD8                   numberof alternate-servers-hint
1     CARD8                   authorization-index
2     a/4                     lengthof alternate-servers-hint
2     (d+q)/4                 lengthof authorization-data
a     LISTofALTERNATESERVER   alternate-servers-hint
d     LISTofBYTE              authorization-data
q                             unused, q=pad(d)
```

If STATUS is Busy or Denied, the protocol stops and the connection is closed. If STATUS is Continue, the client is expected to respond with additional data, to which the server responds with a new status value and more data. This dialog continues until the status is set to Success, or until the server sets STATUS to Busy or Denied and closes the connection:

```
-
4     1+(d+q)/4               length
d     LISTofBYTE              more-authorization-data
q                             unused, q=pad(d)
=
4     2+(d+q)/4               length
2     CARD16                  status
        0                     Success
      1                       Continue
      2                       Busy
      3                       Denied
      2                       unused
d     LISTofBYTE              more-authorization-data
q                             unused, q=pad(d)
```

When STATUS is Success, the protocol resumes with the following sent by the server:

```
4     3+(v+w)/4                length of rest of data
2     CARD16                   maximum-request-length
2     v                        length of vendor string
4     CARD32                   release-number
v     STRING8                  vendor-string
w                              unused, w=pad(v)
```

Once the connection has been established, the client may send the following requests:

NoOp

```
1   0                          major-opcode
1                              unused
2   1                          length
```

ListExtensions

```
1   1                          major-opcode
1                              unused
2   1                          length
=
1   0                          type reply
1   CARD8                      numberof names
2   CARD16                     sequence-number
4   2+(n+p)/4                  length
n   LISTofSTRNAME              names
p                              unused, p=pad(n)
```

QueryExtension

```
1   2                          major-opcode
1   n                          length of name
2   1+(n+p)/4                  length
n   STRING8                    name
p                              unused, p=pad(n)
=
1   0                          type reply
1   BOOL                       present
2   CARD16                     sequence-number
4   5                          length
2   CARD16                     major-version
2   CARD16                     minor-version
1   CARD8                      major-opcode
1   CARD8                      first-event
1   CARD8                      number-events
1   CARD8                      first-error
1   CARD8                      number-errors
3                              unused
```

ListCatalogues

```
1   3                          major-opcode
1                              unused
2   3+(n+p)/4                  length
4   CARD32                     max-names
2   n                          length of pattern
```

```
2                            unused
n   STRING8                  pattern
p                            unused, p=pad(n)
= +
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   4+(n+p)/4                length
4   CARD32                   replies-following-hint
4   CARD32                   numberof catalogue-names
n   LISTofSTRNAME            catalogue-names
p                            unused, p=pad(n)
```

SetCatalogues

```
1   4                        major-opcode
1   CARD8                    numberof catalogue-names
2   1+(n+p)/4                length
n   LISTofSTRNAME            catalogue-names
p                            unused, p=pad(n)
```

GetCatalogues

```
1   5                        major-opcode
1                            unused
2   1                        length
=
1   0                        type reply
1   CARD8                    numberof catalogue-names
2   CARD16                   sequence-number
4   2+(n+p)/4                length
n   LISTofSTRNAME            catalogue-names
p                            unused, p=pad(n)
```

SetEventMask

```
1   6                        major-opcode
1   CARD8                    extension-opcode
2   2                        length
4   EVENTMASK                event-mask
```

GetEventMask

```
1   7                        major-opcode
1   CARD8                    extension-opcode
2   1                        length
=
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   3                        length
4   EVENTMASK                event-mask
```

CreateAC

```
1   8                        major-opcode
1   CARD8                    numberof authorization-protocols
2   2+a/4                    length
```

```
4   ACCESSCONTEXT            ac
a   LISTofAUTH               authorization-protocols
=
1   0                        type reply
1   CARD8                    authorization-index
2   CARD16                   sequence-number
4   3+(d+q)/4                length
2   CARD16                   status
    0 Success
    1 Continue
    2 Busy
    3 Denied
2                            unused
d   LISTofBYTE               authorization-data
q                            unused, q=pad(d)
```

If STATUS is Continue, the client is expected to respond with additional data, to which the server responds with a new status value and more data. This dialog continues until the status is set to Success, Busy, or Denied at which point the request is finished.

```
-
4   1+(d+q)/4                length
d   LISTofBYTE               more-authorization-data
q                            unused, q=pad(d)
=
4   2+(d+q)/4                length
2   CARD16                   status
    0 Success
    1 Continue
    2 Busy
    3 Denied
2                            unused
d   LISTofBYTE               authorization-data
q                            unused, q=pad(d)
```

FreeAC

```
1   9                        major-opcode
1                            unused
2   2                        length
4   ACCESSCONTEXT            ac
```

SetAuthorization

```
1   10                       major-opcode
1                            unused
2   2                        length
4   ACCESSCONTEXT            ac
```

SetResolution

```
1   11                       major-opcode
1   n                        number of resolutions
2   1+(6*n+p)/4              length
6*n LISTofRESOLUTION         resolutions
```

```
p   p=pad(6*n)
```

GetResolution

```
1   12                       major-opcode
1                            unused
2   1                        length
=
1   0                        type reply
1   n                        number of resolutions
2   CARD16                   sequence-number
4   2+(6*n+p)/4              length
6*n LISTofRESOLUTION         resolutions
p   p=pad(6*n)
```

ListFonts

```
1   13                       major-opcode
1                            unused
2   3+(n+p)/4                length
4   CARD32                   max-names
2   n                        length of pattern
2                            unused
n   STRING8                  pattern
p                            unused, p=pad(n)
= +
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   4+(n+p)/4                length
4   CARD32                   replies-following-hint
4   CARD32                   numberof font-names
n   LISTofSTRNAME            font-names
p                            unused, p=pad(n)
```

ListFontsWithXInfo

```
1   14                       major-opcode
1                            unused
2   3+(n+p)/4                length
4   CARD32                   max-names
2   n                        length of pattern
2                            unused
n   STRING8                  pattern
p                            unused, p=pad(n)
= +(except for last in series)
1   0                        type reply
1   n                        length of name
2   CARD16                   sequence-number
4   3+(n+p+f)/4              length
4   CARD32                   replies-hint
f   XFONTINFO                fontinfo
n   STRING8                  name
p                            unused, p=pad(n)
= (last in series)
1   0                        type reply
```

```
1   0                        last-reply indicator
2   CARD16                   sequence-number
4   2                        reply length
```

OpenBitmapFont

```
1   15                       major-opcode
1                            unused
2   4+(n+p)/4                length
4   FONTID                   fontid
4   BITMAPFORMATMASK         format-mask
4   BITMAPFORMAT             format
n   STRNAME                  pattern
p                            unused, p=pad(n)
=
1   0                        type reply
1   BOOL                     otherid-valid
2   CARD16                   sequence-number
4   4                        length
4   FONTID                   otherid
1   BOOL                     cachable
3                            unused
```

QueryXInfo

```
1   16                       major-opcode
1                            unused
2   2                        length
4   FONTID                   fontid
=
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   2+f/4                    length
f   XFONTINFO                fontinfo
p                            unused, p=pad(f)
```

QueryXExtents8

```
1   17                       major-opcode
1   BOOL                     range
2   3+(n+p)/4                length
4   FONTID                   fontid
4   n                        number chars entries
n   STRING8                  chars
p                            unused, p=pad(n)
=
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   3+3*n                    length
4   n                        number of extents
12*n LISTofXCHARINFO         extents
```

QueryXExtents16

```
1   18                       major-opcode
```

```
1   BOOL                     range
2   3+(2*n+p)/4              length
4   FONTID                   fontid
4   n                        number chars entries
2*n                          LISTofCHAR2B chars
p                            unused, p=pad(2*n)
=
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   3+3*n                    length
4   n                        number of extents
12*n LISTofXCHARINFO         extents
```

QueryXBitmaps8

```
1   19                       major-opcode
1   BOOL                     range
2   4+(n+p)/4                length
4   FONTID                   fontid
4   BITMAPFORMAT             format
4   n                        number of chars entries
n   STRING8                  chars
p                            unused, p=pad(n)
= +
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   5+2*n+(m+p)/4            length
4   CARD32                   replies-following-hint
4   n                        number of offsets
4   m                        number of bytes of glyph images
8*n LISTofOFFSET32           offsets
m   LISTofBYTE               glyphimages
p                            unused, p=pad(m)
```

QueryXBitmaps16

```
1   20                       major-opcode
1   BOOL                     range
2   4+(2*n+p)/4              length
4   FONTID                   fontid
4   BITMAPFORMAT             format
4   n                        number of chars entries
2*n LISTofCHAR2B             chars
p                            unused, p=pad(2*n)
=
1   0                        type reply
1                            unused
2   CARD16                   sequence-number
4   5+2*n+(m+p)/4            length
4   CARD32                   replies-following-hint
4   n                        number of offsets
4   m                        number of bytes of glyph images
8*n LISTofOFFSET32           offsets
```

```
m   LISTofBYTE                 glyphimages
p                              unused, p=pad(m)
```

CloseFont

```
1   21                         major-opcode
1                              unused
2   2                          length
4   FONTID                     fontid
```

Errors

Request

```
1   1                          type error
1   0                          Request
2   CARD16                     sequence-number
4   4                          length
4   TIMESTAMP                  timestamp
1   CARD8                      major-opcode
1   CARD8                      minor-opcode
2                              unused
```

Format

```
1   1                          type error
1   1                          Format
2   CARD16                     sequence-number
4   5                          length
4   TIMESTAMP                  timestamp
1   CARD8                      major-opcode
1   CARD8                      minor-opcode
2                              unused
4   BITMAPFORMAT               bad-format
```

Font

```
1   1                          type error
1   2                          Font
2   CARD16                     sequence-number
4   5                          length
4   TIMESTAMP                  timestamp
1   CARD8                      major-opcode
1   CARD8                      minor-opcode
2                              unused
4   FONTID bad-fontid
```

Range

```
1   1                          type error
1   3                          Range
2   CARD16                     sequence-number
4   5                          length
4   TIMESTAMP                  timestamp
1   CARD8                      major-opcode
1   CARD8                      minor-opcode
```

```
2                            unused
4   RANGE                    bad-range
```

EventMask

```
1   1                        type error
1   4                        EventMask
2   CARD16                   sequence-number
4   5                        length
4   TIMESTAMP                timestamp
1   CARD8                    major-opcode
1   CARD8                    minor-opcode
2                            unused
4   EVENTMASK                event-mask
```

AccessContext

```
1   1                        type error
1   5                        AccessContext
2   CARD16                   sequence-number
4   5                        length
4   TIMESTAMP                timestamp
1   CARD8                    major-opcode
1   CARD8                    minor-opcode
2                            unused
4   ACCESSCONTEXT            access context
```

IDChoice

```
1   1                        type error
1   6                        IDChoice
2   CARD16                   sequence-number
4   5                        length
4   TIMESTAMP                timestamp
1   CARD8                    major-opcode
1   CARD8                    minor-opcode
2                            unused
4   FONTID                   bad-fontid
```

Name

```
1   1                        type error
1   7                        Name
2   CARD16                   sequence-number
4   4                        length
4   TIMESTAMP                timestamp
1   CARD8                    major-opcode
1   CARD8                    minor-opcode
2                            unused
```

Resolution

```
1   1                        type error
1   8                        Resolution
2   CARD16                   sequence-number
4   5                        length
4   TIMESTAMP                timestamp
1   CARD8                    major-opcode
```

1	CARD8	minor-opcode
6	RESOLUTION	resolution

Alloc

1	1	type error
1	9	Alloc
2	CARD16	sequence-number
4	4	length
4	TIMESTAMP	timestamp
1	CARD8	major-opcode
1	CARD8	minor-opcode
2		unused

Length

1	1	type error
1	10	Length
2	CARD16	sequence-number
4	5	length
4	TIMESTAMP	timestamp
1	CARD8	major-opcode
1	CARD8	minor-opcode
2		unused
4	CARD32	bad-length

Implementation

1	1	type error
1	11	Implementation
2	CARD16	sequence-number
4	4	length
4	TIMESTAMP	timestamp
1	CARD8	major-opcode
1	CARD8	minor-opcode
2		unused

Events

KeepAlive

1	2	type event
1	0	event KeepAlive
2	CARD16	sequence-number
4	3	length
4	TIMESTAMP	timestamp

CatalogueListNotify

1	2	type event
1	1	event CatalogueListNotify
2	CARD16	sequence-number
4	4	length
4	TIMESTAMP	timestamp
1	BOOL	added
1	BOOL	deleted

```
2                             unused
```

FontListNotify

```
1   2                         type event
1   2                         event FontListNotify
2   CARD16                    sequence-number
4   4                         length
4   TIMESTAMP                 timestamp
1   BOOL                      added
1   BOOL                      deleted
2                             unused
```

Acknowledgements

This document represents the culmination of several years of debate and experiments done under the auspices of the MIT X Consortium font working group. Although this was a group effort, the author remains responsible for any errors or omissions. The protocol presented here was primarily designed by Jim Fulton, Keith Packard, and Bob Scheifler. Special thanks goes to Ned Batchelder, Jim Flowers, and Axel Deininger for their invigorating comments which never failed to make this a better document. Stephen Gildea edited version 2 of this document. Finally, David Lemke deserves great credit for designing and coding the sample implementation.

References

All of the following documents are X Consortium standards available from the X Consortium.

X Window System Protocol Version 11. Robert W. Scheifler.

Adobe System - Bitmap Distribution Format 2.1.

X Consortium. X Logical Font Description Conventions, Version 1.5.

A. Suggested Licensing Policies

The authorization data passed by the client in the initial connection setup information may be used by the font server to implement restrictions on which fonts may be accessed. Furthermore, the font server is free to refuse new connections at any time.

Configuration or management of the license restrictions is outside the scope of the font service protocol and is done in a server-dependent manner. Possible policies might include, but are not limited to, combinations of the following:

- No restrictions - anyone may access any fonts. The server neither refuses any connections nor generates AccessContext errors on any fonts. For environments without specially-licensed fonts, this is sufficient.
- Per-machine - only those clients connecting from a known set of machines are permitted access. The server could get the address of the connection and look in a list of allowed machines.

- Per-user - only a known set of users may access the fonts. The server can use the authorization data (such as a Kerberos ticket or a Secure RPC credential) to verify the identity of the user and then look in a list of allowed users.
- Simultaneous Use - only a certain number of clients may use a given font at any one time. Additional clients would receive AccessContext errors if they attempt to open the font. This is only effective if the initial clients keep the font open for the entire time that it is being used (even if all of the data has been transmitted and is being cached).
- Postage Meter - a particular font may only be accessed a limited number of times before its license must be renewed. Each time the font is opened, the server decrements a counter. When the counter reaches zero, all further attempts to open the font return an AccessContext error.

It should be noted that chaining of font servers (obtaining font data from other font servers) may conflict with certain license policies.

B. Implementation Suggestions

Font server implementations will probably wish to use techniques such as the following to avoid limits on the number of simultaneous connections:

- The initial connection information returned by the font server contains the names of other font servers that may be used as substitutes. A font server may refuse to accept a connection, indicating that the client should try one of the alternatives instead.
- On operating systems that support processing forking, font servers might choose to fork so that the child can continue processing the existing connections and the parent can accept new connections. Such implementations are encouraged to use shared memory so that in-memory font databases can be shared.
- On operating systems that support passing stream file descriptors between processes, cooperating font servers could collect connections in a single process when there are few connections and spread them among several processes as the load increases.
- If a font client is unable to connect to a server (as opposed to having the connection terminated), it should retry for an implementation-dependent length of time (see Xlib's handling of ECONNREFUSED in XConnDis.c).