Copyright (c) 1996 X Consortium

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 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.
1 Introduction

Low Bandwidth X (LBX) is a network-transparent protocol for running X Window System applications over transport channels whose bandwidth and latency are significantly worse than that used in local area networks. It combines a variety of caching and reencoding techniques to reduce the volume of data that must be sent over the wire. It can be used with existing clients by placing a proxy between the clients and server, so that the low bandwidth/high latency communication occurs between the proxy and server.

This extension was designed and implemented by Jim Fulton, David Lemke, Keith Packard, and Dale Tonogai, all of Network Computing Devices (NCD). Chris Kent Kantarjiev (Xerox PARC) participated in early design discussions. Ralph Mor (X Consortium) designed and implemented additional sections. Donna Converse (X Consortium) authored the protocol description and encoding from design notes and the implementation. Ray Tice (X Consortium) resolved the open issues in the design and specification. Bob Scheifler (X Consortium) helped out in many areas.

The extension name is "LBX".

2 Description

The design center for LBX is to use a proxy as an intermediary between the client and server. The proxy reencodes and compresses requests, events, replies and errors, as well as the resulting data stream. Additionally, the proxy can cache information from the server to provide low-latency replies to clients. This reply generation by the proxy is known as short-circuiting. A proxy can handle multiple clients for a given server, but does not prevent clients from connecting directly to the server. The design allows the proxy to multiplex multiple clients into a single data stream to the server.

Much of LBX is implemented as an extension. The compression and reencoding changes can be isolated to the transport and dispatch portions of the server, while short-circuiting requires minor changes to the server’s colormap and property code.

LBX employs several different compression and short-circuiting methods. Use of these methods is negotiable, and in some cases, the algorithm used by a given method is negotiable as well. LBX also provides for negotiation of extensions to LBX.

2.1 Data Flow

The LBX data stream goes through a number of layers:

0. Client requests
1. Read by LBX and potential byte-swapping
2. Request-specific compression
3. Potential byte swapping
4. Multiplexing of client request streams
5. Delta replacement
6. Stream compression

Transport

6. Stream decompression
5. Delta substitution
4. Demultiplexing of client request streams
3. Potential byte swapping
2. Reencoding
1. Request processing

The reverse process occurs with X server replies, events, and errors.

2.2 Tags

Tags are used to support caching of large data items that are expected to be queried multiple times. Such things as the keyboard map and font metrics are often requested by multiple clients. Rather than send the data each time, the first time the data is sent it includes a tag. The proxy saves this data, so that subsequent requests can send only the tag to refer to that same data. The different types of tags are used for connection information, keyboard maps, modifier maps, fonts information and properties.

Tag usage is negotiated as a boolean in the LbxStartProxy message. The proxy controls how many tags are stored in the proxy. The server may wish to observe the proxy's InvalidateTag behavior to limit how many tags are cached at any one time. Tagged data is not shared across types of tags, but the number space used for the tag ids is. The tag ids are generated by the server.

The X server keeps track of what tags are known to the proxy. The proxy can invalidate a tag if no tag bearing replies of that type are pending. The proxy sends an LbxInvalidateTag message to release the tagged data. The proxy must not invalidate connection tags unless instructed to do so by the server.

If the server wishes to discard tagged data, it must either have received an LbxInvalidateTag request from the proxy or send an LbxInvalidateTag event to the proxy for that tag.

2.2.1 Tag Substitution in Requests

Many substitution requests have a tag field, followed by fields marked optional. For these requests, if the optional fields are present, the data in them is stored in the indicated tag, unless the tag is 0. If the optional fields are absent, the tag field indicates the tag that contains the data for the "optional" fields.

2.2.2 Property Tags

Property data makes special use of tags. A common use of properties is for inter-client communication. If both clients use the proxy, it is wasteful to send the data to the server and then back, when the server may never need it. LbxChangeProperty request does the same work as the core ChangeProperty request, but it does not send the data. The reply to this request contains a tag id corresponding to the data. If the property information is used locally, the server responds to LbxGetProperty with the tag, and the property data need never be sent to the server. If the server does require the data, it can issue an LbxQueryTag message. The proxy can also send the data on at any time if it judges it appropriate (i.e., when the wire goes idle). Since the proxy owns the property data, it must not invalidate the tag before sending the data back to the server via an LbxTagData request.

2.3 Short-circuiting

Short-circuiting is used to handle constant data. This includes atoms, color name/RGB mappings, and AllocColor calls. Atoms and color name/RGB mappings stay constant for the life of the server. AllocColor replies are constant for each colormap. Short-circuiting replaces round-trip requests with one-way requests, and can sometimes use one in place of many.
Atoms are used heavily for ICCCM communication. Once the proxy knows the string to atom mapping, it has no need to send subsequent requests for this atom to the server.

Colorname/RGB mappings are constant, so once the proxy sees the response from LookupColor, it need not forward any subsequent requests.

Clients often use the same color cells, so once a read-only color allocation has occurred, the proxy knows what RGB values should be returned to the client. The proxy doesn’t need to forward any AllocColor requests it can resolve, but it must tell the server to modify the color cell’s reference count LbxIncrementPixel is used to support this.

For all three classes of short-circuiting, the proxy must still tell the server a request has occurred, so that the request sequence numbers stay in sync. This is done with LbxModifySequence.

Sequence numbers cause the major complication with short-circuiting. X guarantees that any replies, events or errors generated by a previous request will be sent before those of a later request. This means that any requests that can be handled by the proxy must have their reply sent after any previous events or errors.

If a proxy’s applications do not require strict adherence to the X protocol ordering of errors or events, a proxy might provide further optimization by avoiding the overhead of maintaining this ordering, however, the resulting protocol is not strictly X11 compliant.

### 2.4 Graphics Re-encoding

The LBX proxy attempts to reencode PolyPoint, PolyLine, PolySegment, PolyRectangle, PolyArc, FillPoly, PolyFillRectangle, PolyFillArc, CopyArea, CopyPlane, PolyText8, PolyText16, ImageText8, and ImageText16 requests. If the request can be reencoded, it may be replaced by an equivalent LBX form of the request. The requests are reencoded by attempting to reduce 2-byte coordinate, length, width and angle fields to 1 byte. Where applicable, the coordinate mode is also converted to Previous to improve the compressibility of the resulting data. In image requests, the image data may also be compressed.

### 2.5 Motion events

To prevent clogging the wire with MotionNotify events, the server and proxy work together to control the number of events on the wire. This is done with the LbxAllowMotion request. The request adds an amount to an allowed motion count in the server, which is kept on a per-proxy basis. Every motion notify event sent to the proxy decrements the allowed motion counter. If the allowed motion count is less than or equal to zero, motion events not required by the X protocol definition are not sent to the proxy. The allowed motion counter has a minimum value of -2^31.

### 2.6 Event Squishing

In the core protocol, all events are padded as needed to be 32 bytes long. The LBX extension reduces traffic by removing padding at the end of events, and implying the event length from its type. This is known as squishing.
2.7 Master Client

When the initial X connection between the proxy and the server is converted to LBX mode, the proxy itself becomes the master client. New client requests and some tag messages are sent in the context of the master client.

2.8 Multiplexing of Clients

The LBX proxy multiplexes the data streams of all its clients into one stream, and then splits them apart again when they are received. The LbxSwitch message is used to tell each end which client is using the wire at the time.

The server should process delta requests in the order that they appear on the LBX connection. If the server does not maintain the interclient request order for requests sent by the proxy, it must still obey the semantics implied by the interclient request order so that the delta cache functions correctly.

The server can affect the multiplexing of clients by the proxy using the LbxListenToOne and LbxListenToAll messages. This is useful during grabs, since the master connection can not be blocked during grabs like other clients. The proxy is responsible for tracking server grabs issued by its clients so that the proxy can multiplex the client streams in an order executable by the server.

Replies must be ordered in the multiplexed data stream from the server to the proxy such that the reply carrying tagged data precedes replies that refer to that tagged data.

2.9 Swapping

Swapping is handled as with any X extension, with one caveat. Since a proxy can be supporting clients with different byte orders, and they all share the same wire, the length fields of all messages between the server and proxy are expressed in the proxy byte order. This prevents any problems with length computation that may occur when clients are switched.

2.10 Delta cache

LBX takes advantage of the fact that an X message may be very similar to one that has been previously sent. For example, a KeyPress event may differ from a previous KeyPress event in just a few bytes. By sending just the bytes that differ (or “deltas”), the number of bytes sent over the wire can be substantially reduced. Delta compaction is used on requests being sent by the proxy as well as on replies and events being sent by the server.

The server and the proxy each keep per-proxy request and response caches. The response cache contains events, errors and replies. All messages are saved in the appropriate delta cache if they are of an appropriate type and more than 8 bytes long but fit within the delta cache. The number of entries in the delta cache and the maximum saved message size are negotiated in the LbxStartProxy request.

The LBX requests that are never stored in the request delta cache are the LbxQueryVersion, LbxStartProxy, LbxSwitch, LbxNewClient, LbxAllowMotion, LbxDelta, LbxQueryExtension, LbxPutImage, LbxGetImage, LbxBeginLargeRequest, LbxLargeRequestData, LbxEndLargeRequest and LbxInternAtoms requests. The responses that are never stored in the response cache are LbxSwitchEvent and LbxDeltaResponse. The message carried by a delta message is also cached, if it meets the other requirements. Messages after the LbxStartProxy request are
C Library Interfaces

The C Library routines for LBX are in the Xext library. The prototypes are located in a file named "XLbx.h".

3.1 Application Library Interfaces

In a proxy environment, applications do not need to call these routines to take advantage of LBX. Clients can, however, obtain information about the LBX extension to the server using this interface. Use of this routine may be altered when connected through a proxy, as described in “C Library Interfaces” on page 7.

3.1.1 XLbxQueryVersion

To determine the version of LBX supported by the X server, call XLbxQueryVersion.

Bool XLbxQueryVersion(display, major_version_return, minor_version_return)

Display * display;
int * major_version_return;
int * minor_version_return;

display Specifies the connection to the X server.
major_version_return Returns the extension major version number.
minor_version_return Returns the extension minor version number.
The XLbxQueryVersion function determines if the LBX extension is present. If the extension is not present, XLbxQueryVersion returns False; otherwise, it returns True. If the extension is present, XLbxQueryVersion returns the major and minor version numbers of the extension as supported by the X server.

3.2 Proxy Library Interfaces

The following interfaces are intended for use by the proxy.

3.2.1 XLbxQueryExtension

To determine the dynamically assigned codes for the extension, use the Xlib function XQueryExtension or the LBX function XLbxQueryExtension.

```
Bool XLbxQueryExtension(display, major_opcode_return, first_event_return, first_error_return)
```

- **display** Specifes the connection to the X server.
- **major_opcode_return** Returns the major opcode.
- **first_event_return** Returns the first event code.
- **first_error_return** Returns the first error code.

The XLbxQueryExtension function determines if the LBX extension is present. If the extension is not present, XLbxQueryExtension returns False; otherwise, it returns True. If the extension is present, XLbxQueryExtension returns the major opcode for the extension to major_opcode_return, the base event type code to first_event_return, and the base error code to first_error_return; otherwise, the return values are undefined.

3.2.2 XLbxGetEventBase

To determine the base event type code, use the Xlib function XQueryExtension or the LBX function XLbxGetEventBase.

```
int XLbxGetEventBase(display)
```

- **display** Specifes the connection to the X server.

The XLbxGetEventBase function returns the base event type code if the extension is present; otherwise, it returns -1.

4 Protocol

4.1 Syntactic Conventions and Common Types

Please refer to the X Window System Protocol specification, as this document uses the syntactic conventions established there and references types defined there.

The following additional types are defined by this extension:

DIFFITEM

```
1 CARD8 offset
```
Protocol

1 CARD8 diff

LBXANGLE: CARD8 or 2 BYTE

where (in order of precedence):

- \(0 \leq\text{in} \leq A(95)\) \&\& !(\text{in} \% A(5)) \rightarrow out = 0x5a + (\text{in} / A(5))
- \(A(105) \leq\text{in} \leq A(360)\) \&\& !(\text{in} \% A(15)) \rightarrow out = 0x67 + (\text{in} / A(15))
- \(-A(100) \leq\text{in} \leq -A(5)\) \&\& !(\text{in} \% A(5)) \rightarrow out = 0xa6 + (\text{in} / A(5))
- \(-A(360) <\text{in} \leq A(360)\) \rightarrow out[0] = \text{in} >> 8; out[1] = \text{in}

LBXARC:

[x, y: LBXINT16,
width, height: LBXCARD16,
angle1, angle2: LBXANGLE]

Within a list of arcs, after the first arc, x and y are relative to the corresponding fields of the prior arc.

LBXCARD16: CARD8 or 2 BYTE

where:

- \(0x0000 \leq\text{in} < 0x00F0\) \rightarrow out[0] = xF0 | ((in - 0xF0) >> 8) \hspace{1cm} out[1] = in - 0xF0
- \(0x00F0 \leq\text{in} < 0x10F0\) \rightarrow out[0] = 0xF0 | ((in - 0xF0) >> 8) \hspace{1cm} out[1] = in - 0xF0

LBXGCANDDRAWENT

[ gc-cache-index, drawable-cache-index: CARD4 ]

LBXGCANDDRAWUPDATE

drawable: DRAWABLE
\hspace{1cm} /* present only if drawable-cache-index == 0 */

gc: GC
\hspace{1cm} /* present only if gc-cache-index == 0 */

LBXGCANDDRAWABLE

cache-entries: LBXGCANDDRAWENT
updates: LBXGCANDDRAWUPDATE

LBXINT16: INT8 or 2 BYTE

where:

- \(0xFE00 \leq\text{in} < 0x0000\) \rightarrow out[0] = 0xF0 | (\text{in} + 0x1000) >> 8 \hspace{1cm} out[1] = in + 0x1000
- \(0xFF90 \leq\text{in} < 0x0080\) \rightarrow out[0] = 0x80 | (\text{in} - 0x80) >> 8 \hspace{1cm} out[1] = in - 0x80
- \(0x0080 \leq\text{in} < 0x0880\) \rightarrow out[0] = 0x80 | (\text{in} - 0x80) >> 8 \hspace{1cm} out[1] = in - 0x80
- \(0x0000 \leq\text{in} < 0xFF90\) \rightarrow out[0] = 0xF0 | (\text{in} + 0x70) >> 8 \hspace{1cm} out[1] = in + 0x70

LBXPINT16: CARD8 or 2 BYTE

\hspace{1cm} /* for usually positive numbers */

where:

- \(0xFE00 \leq\text{in} < 0x0000\) \rightarrow out[0] = 0xF0 | (\text{in} + 0x1000) >> 8 \hspace{1cm} out[1] = in + 0x1000
- \(0x0000 \leq\text{in} < 0xFF90\) \rightarrow out[0] = 0xF0 | (\text{in} + 0x70) >> 8 \hspace{1cm} out[1] = in + 0x70
LBX

0x0000 <= in < 0x00F0
CARD8
0x00F0 <= in < 0x0EF0
out[0] = 0xF0 | ((in - 0xF0) >>8)
out[1] = in - 0xF0

LBXPOINT: [x, y: LBXINT16]
Within a list of points, after the first rectangle, x and y are relative to the corresponding fields of the prior point.

LBXRECTANGLE:
[x, y: LBXINT16,
width, height: LBXCARD16]
Within a list of rectangles, after the first rectangle, x and y are relative to the corresponding fields of the prior rectangle.

MASK: CARD8

4.2 Errors
As with the X11 protocol, when a request terminates with an error, the request has no side effects (that is, there is no partial execution).
There is one error, LbxClient. This error indicates that the client field of an LBX request was invalid, or that the proxy’s connection was in an invalid state for a start or stop proxy request.

4.3 Requests
There is one request that is expected to be used only by the client: LbxQueryVersion
There is one request that is expected to be used by the client or the proxy: LbxQueryExtension.
The following requests are expected to be used only by the proxy, and are instigated by the proxy: Lbx-StartProxy, LbxStopProxy, LbxNewClient, LbxSwitch, LbxCloseClient, LbxModifySequence, LbxAllowMotion, LbxInvalidateTag, LbxTagData and LbxQueryTag.
All other requests are sent by the proxy to the LBX server and are instigated by reception of an X request from the client. They replace the X request.

4.3.1 Requests Initiated by the Proxy or by the Client
LbxQueryVersion
→
    majorVersion: CARD16
    minorVersion: CARD16
This request returns the major and minor version numbers of the LBX protocol.
The encoding of this request is on page 32.
### 4.3.2 Requests Initiated or Substituted by the Proxy

**LbxQueryExtension**

\[\text{nbytes}: \text{CARD32} \]
\[\text{name}: \text{STRING8} \]

\[\text{num-requests}: \text{CARD8} \]
\[\text{present}: \text{BOOL} \]
\[\text{major-opcode}: \text{CARD8} \]
\[\text{rst-event}: \text{CARD8} \]
\[\text{rst-error}: \text{CARD8} \]
\[\text{reply-mask}: \text{LISTofMASK} /* \text{optional} */ \]
\[\text{event-mask}: \text{LISTofMASK} /* \text{optional} */ \]

**Errors:** Alloc

This request is identical to the QueryExtension request, with an additional field, and two optional additional fields. When the client issues a QueryExtension request, the proxy will substitute an LbxQueryExtension request.

This request determines if the named extension is present. If so, the major opcode for the extension is returned, if it has one. Otherwise, zero is returned. Any minor opcode and the request formats are specific to the extension. If the extension involves additional event types, the base event type code is returned. Otherwise, zero is returned. The format of events is specific to the extension. If the extension involves additional error codes, the base error code is returned. Otherwise, zero is returned. The format of additional data in the errors is specific to the extension.

In addition, the number of requests defined by the named extension is returned. If the number of requests is nonzero, and if the information is available, reply-mask and event-mask will be included in the reply. The reply-mask represents a bit-wise one-to-one correspondence with the extension requests. The least significant bit corresponds to the first request, and the next bit corresponds to the next request, and so on. Each element in the list contains eight meaningful bits, except for the last element, which contains eight or fewer meaningful bits. Unused bits are not guaranteed to be zero. The bit corresponding to a request is set if the request could generate a reply, otherwise it is zero. In the same way, the event-mask represents a bit-wise one-to-one correspondence with the extension requests. A bit is set if the corresponding request could result in the generation of one or more extension or X11 events. If reply-mask is present in the reply, event-mask will also be present.

The encoding of this request is on page 43.

### 4.3.3 Control Requests Initiated by the Proxy

**LbxStartProxy**

\[\text{options}: \text{LISTofOPTION} \]

\[\text{choices}: \text{LISTofCHOICE} \]

**Errors:** LbxClient, Alloc
where:

OPTION
[opcode: CARD8,
len: OPTLEN,
option: (See Table 1, “StartProxy Options,” on page12)]

CHOICE
[opcode: CARD8,
len: OPTLEN,
choice: (See Table 1, “StartProxy Options,” on page12)]

<table>
<thead>
<tr>
<th>Table 1. StartProxy Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Option</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>delta-proxy</td>
</tr>
<tr>
<td>delta-server</td>
</tr>
<tr>
<td>stream-comp</td>
</tr>
<tr>
<td>bitmap-comp</td>
</tr>
<tr>
<td>pixmap-comp</td>
</tr>
<tr>
<td>use-squish</td>
</tr>
<tr>
<td>use-tags</td>
</tr>
<tr>
<td>colormap</td>
</tr>
<tr>
<td>extension</td>
</tr>
</tbody>
</table>

DELTAOPT [minN, maxN, prefN: CARD8
minMaxMsgLen, maxMaxMsgLen, prefMaxMsgLen: CARD8]

DELTACHoice [entries, maxlen: CARD8]

INDEXEDCHOICE [index: CARD8,
data: LISTOfString8]

INDEXEDOPT [index: STRING8,
detail: LISTOfString8]

NAMEDOPT [name: STRING8,
detail: LISTOfString8]

OPTLEN 1 or 3 CARD8

where:

(0 < in <= 0xFF): out = in
(0 <= in<= 0xFFFF): out[0] = 0; out[1] = in >> 8; out[2] = in& 0xFF;

PIXMAPMETHOD [name: STRING8,
format-mask: BITMASK,
depths: LISTOfCARD8]

PIXMAPCHOICE [index, opcode: CARD8,
format-mask: BITMASK,
depths: LISTOfCARD8]

This request negotiates LBX protocol options, and switches the proxy-server connection from X11 protocol to LBX protocol.

The proxy gives the preferred protocol options in the request. The server chooses from the given options and informs the proxy which to use. The options may be listed in any order, and the proxy may choose which options to negotiate. If an option is not successfully negotiated, the default is used.

The server delta cache and proxy delta caches can be configured for number of entries, and the length of entries. (See “Delta cache” on page6 for details.) The delta caches are configured using the delta-server and...
delta-proxy options. To configure a cache, the proxy sends the minimum, maximum and preferred values for the number of cache entries, \((\text{minN}, \text{maxN}, \text{prefN})\), and the length of the cache entries, \((\text{minMaxMsgLen}, \text{maxMaxMsgLen}, \text{prefMaxMsgLen})\). The server’s reply fields \(\text{entries}\) and \(\text{maxlen}\) contains the values to use. These values must be within the ranges specified by the proxy. The server may also specify an \(\text{entries}\) value of 0 to disable delta caching. The cache entry lengths are specified in units of 4 bytes.

The stream compression algorithm is selected using the \textit{stream-comp} option. (Stream compression is described in “Stream Compression” on page 7.) Each algorithm has a name that follows the naming conventions in “Algorithm Naming” on page 31. To negotiate using the stream-comp option, the proxy lists its available compressors. For each candidate algorithm, the proxy sends the name in the \textit{name} field, and uses the \textit{detail} field to send any additional data specific to each compression algorithm. The reply contains a 0-based index into the list of algorithms to indicate which algorithm to use, followed by data specific to that algorithm.

Bitmap compression is negotiated using the \textit{bitmap-comp} option. The proxy sends a list of names of available algorithms, and the server reply lists the algorithms to use. For each bitmap algorithm in the reply, a 0-based index into the list of bitmap algorithms indicates the algorithm, and the \textit{opcode} field gives the value for use in requests. The algorithm names follow the conventions in “Algorithm Naming” on page 31.

Pixmap compression is negotiated using the \textit{pixmap-comp} option. The proxy sends a list of available algorithms. For each algorithm, the list includes the name, a bitmask of supported formats, and a list of depths that the format supports. The server reply lists the algorithms to use. For each pixmap algorithm in the reply, the reply contains a 0-based index into the list of pixmap algorithms, the opcode to use in requests when referring to this algorithm, a mask of valid formats, and a list of valid depths. Algorithm names follow the conventions in “Algorithm Naming” on page 31.

Squishing is negotiated using the \textit{use-squish} option. If the proxy desires squishing, it sends a true value. The reply from the server indicates whether to do squishing, and will indicate squishing only if \textit{use-squish} is set to true in the request.

Tag caching, described in “Tags” on page 4, is negotiated using the \textit{use-tag} option. If the proxy desires tag caching, it sends a true value. The reply from the server indicates whether to do tag caching, and will demand caching only if \textit{use-tag} is set to true in the request.

The colormap option is used to negotiate what color matching algorithm will be used by the proxy when the proxy uses the \textit{LbxAllocColor} request to allocate pixels in a grabbed colormap. To negotiate using the colormap option, the proxy lists the names of available colormap algorithms. The choice in the reply contains a 0-based index into the list of algorithms to indicate which algorithm to use, followed by data specific to that algorithm. If no colormap algorithm is successfully negotiated, then the \textit{LbxAllocColor}, \textit{LbxGrabCmap}, and \textit{LbxReleaseCmap} requests will not be used.

The extension option is used to control extensions to LBX. These extensions may, for example, enable other types of compression. To negotiate an extension, the name of the extension is sent, followed by any data specific to that extension. The extension name follows the conventions in “Algorithm Naming” on page 31. The extension option may occur multiple times in the start proxy message, since multiple extensions can be negotiated. The reply to an extension option contains the zero-based index of the extension option, as counted in the \textit{LbxStartProxy} message. This index is followed by extension-specific information. The server does not respond to extensions it does not recognize.

An \textit{LbxClient} error is returned when a client which is already communicating through an LBX proxy to the X server sends a \textit{LbxStartProxy} request.

The encoding for this request is on page 33.
**LbxStopProxy**

Errors: LbxClient

This request terminates the connection between the proxy and X server, and terminates any clients connected through the proxy.

The encoding for this request is on page 35.

An LbxClient error is returned if the requesting client is not an LBX proxy.

**LbxNewClient**

byte-order: CARD8
client-id: CARD32
protocol-major-version: CARD16
protocol-minor-version: CARD16
authorization-protocol-name: STRING8
authorization-protocol-data: STRING8

→ Core X reply (if connection is rejected)

OR

success: BOOL
change-type: {NoDeltas, NormalClientDeltas, AppGroupDeltas}
protocol-major-version: CARD16
protocol-minor-version: CARD16
tag-id: CARD32
length: CARD16
connection-data: CONINFO or CONDIF or CONDIFROOT

where:

<table>
<thead>
<tr>
<th>CONINFO:</th>
<th>(the &quot;additional data&quot; portion of the core connection reply for successes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONDIF:</td>
<td>[resource-id-base: CARD32, root-input-masks: LISTofSETofEVENT]</td>
</tr>
</tbody>
</table>

Errors: LbxClient, Alloc

This request, which is sent by the proxy over the control connection, creates a new virtual connection to the server.

Much of the information in the LbxNewClient request and reply is identical to the connection setup and reply information in the core X protocol.
Protocol

For the LbxNewClient request, the field unique to LBX is client-id. For the LbxNewClient reply, tag-id and change-type are fields unique to LBX, and the contents of connection-data may be different in LBX from the core X protocol (see below).

The proxy assigns each virtual connection a unique identifier using the client-id field in the LbxNewClient request. This client-id is used in the LBX protocol to specify the current client (see the LbxSwitch request and the LbxSwitchEvent). client-id 0 is reserved for the proxy control connection. An LbxClient error will result if the LbxNewClient request contains a client-id of 0 or an already in use client-id.

If the server rejects this new virtual connection, the server sends a core X connection failure reply to the proxy. The current version of LBX does not support the return of an Authenticate reply.

If the change-type field is set to NoDeltas, then connection-data is sent using the CONINFO structure, which is identical to the additional data of the core connection reply. If the tag-id is non-zero, then the connection-data is stored by the proxy using this tag value. Tagged connection data must be stored by the proxy, and can not be invalidated by the proxy until an LbxInvalidateTag event is received for that tag.

When the change-type field is not set to NoDeltas, then connection data is sent as changes against connection information previously sent to the proxy. The tag-id field, if non-zero, has the tag of the previously sent data to apply the changes to. A zero tag-id indicates that the changes are with respect to the connection information sent when the proxy connected to the server.

If the change-type field is set to NormalClientDeltas, then connection-data is sent using the CONDIF structure. The values in the CONDIF structure are substituted for the identically named fields of the connection information for the new connection.

If the change-type field is set to AppGroupDeltas, then connection-data is sent using the CONDIFROOT structure. The root, root-visual, and default-colormap fields, when nonzero, are substituted for the corresponding fields in the reference connection information. The white-pixel and black-pixel fields are substituted only when the default-colormap field of the reply is non-zero. When default-colormap field of the reply is zero, so are white-pixel and black-pixel. The first entry in the root-input-masks field is the current-input-mask for the default root window. The remaining entries in root-input-masks are input masks for non-video screens, as defined by the X Print Extension. The number of non-video screens is one less than the number of entries in root-input-masks. These screens are at the end of screen list in the reference connection information.

The encoding for this request is on page 35.

LbxCloseClient

client: CARD32

Errors: LbxClient

This requests the server to close down the connection represented by the specified proxy’s client identifier. If the specified client wasn’t previously registered with the server by a LbxNewClient request, the server will send the LbxClient error.

The encoding for this request is on page 36.

LbxSwitch

client: CARD32
Errors: LbxClient

This request causes the X server to treat subsequent requests as being from a connection to the X server represented by the specified client identifier.

If the client making the request is not the proxy, or if the client identifier sent in the request was not previously sent in a LbxNewClient request, an LbxClient error is returned.

The encoding for this request is on page 35.

LbxSync

→

The sync request causes the server to send a reply when all requests before the sync request have been processed.

The encoding for this client is on page 48.

LbxModifySequence

  adjust: CARD32

Errors: None

This request advances the sequence number of the virtual client connection by the specified amount. The proxy sends the LbxModifySequence request to the server when it replies to a client request without forwarding the client request on to the X server.

The encoding for this client is on page 36.

LbxAllowMotion

  num: CARD32

Errors: None

This request controls the delivery of optional motion notify events, as described in “Motion events” on page 5. The num field specifies an increase in the allowed number of motion notify events sent.

The encoding for this request is on page 36.

LbxInvalidateTag

  tag: CARD32

The LBX proxy sends this notification to the X server when it refuses to store tagged data, or when it releases tagged data which was previously stored and which was not invalidated by a notification from the X server.

The encoding for this request is on page 37.

LbxTagData
Protocol

**Protocol 17**

- **tag**: CARD32
- **real-length**: CARD32
- **data**: LISTofBYTE

This request specifies the data associated with a previously assigned tag. It is sent in two circumstances: in response to receiving a `SendTagDataEvent`, and spontaneously, when the proxy must rely on the server to store data which was not previously received from the server. The data is carried in the byte order and structure as would have originally been sent in the core protocol request.

The encoding for this request is on page 41.

**LbxGrabCmap**

- **cmap**: Colormap

->

- **smart-grab**: BOOL /* optional */
- **large-pixel**: BOOL /* optional */
- **auto-release**: BOOL /* optional */
- **three-channels**: BOOL /* optional */
- **bits-per-rgb**: CARD4 /* optional */
- **cells**: LISTofCHAN /* optional */

where:

- **CHAN**: LISTofLBXPIXEL
- **LBXPIXEL**: PIXELPRIVATE or PIXELPRIVATERANGE or PIXELALLOC or PIXELALLOCRANGE
- **PIXEL**: CARD8 or CARD16
- **PIXELPRIVATE**: [ pixel: PIXEL ]
- **PIXELPRIVATERANGE**: [ fi rst-pixel, last-pixel: PIXEL]
- **PIXELALLOC**: [ pixel: PIXEL,
  color: COLORSINGLE or COLOTRIPLE]
- **PIXELALLOCRANGE**: [ fi rst-pixel, last-pixel: PIXEL,
  colors: LISTofCOLORSINGLE or LISTofCOLOTRIPLE]
- **COLORSINGLE**: [ value: CARD8 or CARD16 ]
- **COLOTRIPLE**: [ r, g, b: COLORSINGLE]

**Errors**: Colormap

This request asks the server for control of allocating new colormap cells in the specified colormap. The server grants control by replying to this request. If no changes have occurred since the last time this proxy grabbed this colormap, then the **smart-grab** field of the reply is set to true, and the optional fields are not sent. Otherwise, the current contents of the colormap are placed in the reply, as described later in this section.

Once the proxy has received the reply, it can use the `LbxAllocColor` request to allocate new colormap cells without the performance penalty of round trips. The proxy is still permitted to use the normal colormap and `LbxIncrementPixel` requests while the colormap is grabbed. The grab is valid across all virtual connections of the proxy.

The `LbxGrabCmap` request is limited to colormaps for the visual types negotiated as part of the colormap algorithm negotiation in the start proxy request at connection setup.

December 21, 1996

The server and other proxies may not allocate new colormap cells in the colormap while the colormap is
grabbed by this proxy. If the server or another proxy needs to allocate new colormap cells, the server sends a
LbxReleaseCmap event to the proxy holding the grab, which then issues an LbxReleaseCmap request.

The server and other proxies may free colormap cells in a colormap grabbed by a proxy. The server will send
an LbxFreeCells event to the proxy that currently has the colormap grabbed when the cell reference
count reaches 0.

If the colormap is a of a static visual type, such as StaticGray, StaticColor, GrayScale, or
TrueColor, then the proxy’s grab is immediately released by the server, and the proxy must use LbxIn-
crementPixel requests in place of LbxAllocColor requests for this colormap.

If the cmap field does not refer to a valid colormap or the colormap is already grabbed by this proxy then a
Colormap error is generated.

The reply describes the contents of the colormap via several arguments and a descriptive list containing one
or three channels, with each channel describing allocations in the colormap.

The large-pixel argument, if True, specifies that PIXEL indices will be listed as CARD16 quantities instead
of CARD8. The auto-release field, if True, indicates that this colormap is of a static visual type and the
proxy’s grab is immediately released by the server.

If three-channels is False, a single channel is enclosed and color values are described using COLORTRI-
PLE, which has fields for red, green and blue. A single channel is used when the visual type is not Direct-
Color or TrueColor.

If three-channels is True, separate red, green and blue channel lists are enclosed, for describing a Direct-
Color or TrueColor colormap. Color values for entries in each channel are sent using COLORSINGLE
and the corresponding PIXEL value refers to the RGB subfield of the current channel, as defined by the cor-
responding red-mask, green-mask and blue-mask of the visual.

The bits-per-rgb value is one less than the bits-per-rgb-field of the visual that the colormap belongs to.
If the value is 7 or less, then COLORSINGLE values in the descriptive list are sent using CARD8 fields.
Otherwise these values are sent using CARD16 fields.

The list describing current colormap allocations contains entries of the following types:

An LBXPIXELPRIVATE entry indicates that the pixel in the pixel field is unavailable for allocation.

An LBXPIXELPRIVATERANGE entry indicates that a contiguous range of pixels are unavailable for allo-
cation. The range is first-pixel to last-pixel, and includes last-pixel.

An LBXPIXELALLOC entry indicates that the pixel in the pixel field is allocated as a read-only pixel. The
color field carries the color information of the pixel.

An LBXPIXELALLOC RANGE entry indicates that a contiguous range of pixels are allocated as read-only.
The range starts first-pixel to last-pixel, and includes last-pixel. These fields are followed by a list of COL-
ORSINGLE or COLORTRIPLE, depending on the value of three-channels.

A NEXTCHANNEL entry indicates that the next channel of the colormap will be described.

A LISTEND entry indicates the end of the colormap description.

All pixels not described in the reply are unallocated.
The encoding for this request is on page 46.

**LbxReleaseCmap**

\[ cmap: \text{Colormap} \]

This request releases the specified grabbed colormap. If the `cmap` field does not refer to a colormap, a `Bad- Colormap` error is produced.

The proxy must remember the state of the colormap when the `LbxReleaseCmap` request is issued if this proxy may at some future time issue another `LbxGrabCmap` request on this colormap before the state of the colormap changes.

The encoding for this request is on page 48.

**LbxInternAtoms**

\[ count: \text{CARD16} \]
\[ names: \text{LISTofSTRING8} \]
\[ \rightarrow \]
\[ atoms: \text{LISTofATOM} \]

Errors: Alloc

This request allows the proxy to intern a group of atoms in a single round trip. The server will create any atoms that do not exist.

The encoding for this request is on page 45.

### 4.3.4 Substitution Requests

**LbxAllocColor**

\[ cmap: \text{Colormap} \]
\[ pixel: \text{CARD32} \]
\[ red, green, blue: \text{CARD16} \]

This request is sent by a proxy that has given colormap grabbed to allocate a new read-only cell in the colormap. The proxy may substitute this request for the core `AllocColor` and `AllocNamedColor` requests.

The `pixel` field identifies the colormap cell to allocate. The `red`, `green`, and `blue` fields are the hardware specific color values of the corresponding fields of the core `AllocColor` request. The mapping to hardware specific colormap values by the proxy is performed using the color algorithm negotiated by `LbxStartProxy`.

For colormaps of static visual types, the `LbxIncrementPixel` request is used instead of LBX `AllocColor`.

If the `cmap` field does not identify a grabbed colormap then a `BadAccess` error is produced. If the `pixel` field refers to a read-write entry or the `pixel` field refers to a pixel outside of the range of this colormap, a `BadAlloc` error is produced.
The encoding for this request is on page 48.

**LbxIncrementPixel**

```plaintext
cmap: COLORMAP
pixel: CARD32
```

Errors: None

This request replaces the AllocColor request for read-only pixels currently allocated for the current client. If the visual type of the colormap is of a static type, this request may be used on currently unallocated pixels. The colormap is not required to be grabbed to use this request.

The encoding for this request is on page 36.

**LbxDelta**

```plaintext
count: CARD8
cache-index: CARD8
diffs: LISTofDIFFITEM
```

This request contains a minimal amount of information relative to a similar prior request. The information is in the form of a difference comparison to a prior request. The prior request is specified by an index to a cache, independently maintained by both the proxy and the server.

The encoding for this request is on page 36.

**LbxGetModifierMapping**

```plaintext
keyspermod: CARD8
tag: CARD32
keycodes: LISTofKEYCODE /* optional */
```

This request is identical to the core GetModifierMapping request, with the addition of a tag being returned in the reply. See "Tag Substitution in Requests" on page 4 for a description of the tag field and optional fields.

The encoding for this request is on page 37.

**LbxGetKeyboardMapping**

```plaintext
fi rstKeyCode KEYCODE
count: CARD8
```

```plaintext
keysperkeycode: CARD8
tag: CARD32
keysyms: LISTofKEYSYM /* optional */
```

Errors: Value
This request is identical to the X \texttt{GetKeyboardMapping} protocol request, with the addition that a tag is returned in the reply. See “Tag Substitution in Requests” on page 4 for a description of the tag field and optional fields.

The encoding for this request is on page 39.

\textbf{\texttt{LbxGetWinAttrAndGeom}}

\[ \text{window: WINDOW} \]

\[ \rightarrow \]

\[ \text{visual: VISUALID} \]
\[ \text{class: \{InputOutput, InputOnly\}} \]
\[ \text{bit-gravity: BITGRAVITY} \]
\[ \text{win-gravity: WINGRAVITY} \]
\[ \text{backing-store: \{NotUseful, WhenMapped, Always\}} \]
\[ \text{backing-planes: CARD32} \]
\[ \text{backing-pixel: CARD32} \]
\[ \text{save-under: BOOL} \]
\[ \text{colormap: COLORMAP or None} \]
\[ \text{map-is-installed: BOOL} \]
\[ \text{map-state: \{Unmapped, Unviewable, Viewable\}} \]
\[ \text{all-event-masks, your-event-mask: SETofEVENT} \]
\[ \text{do-not-propagate-mask: SETofDEVICEEVENT} \]
\[ \text{override-redirect: BOOL} \]
\[ \text{root: WINDOW} \]
\[ \text{depth: CARD8} \]
\[ \text{x, y: INT16} \]
\[ \text{width, height, border-width: CARD16} \]

\textbf{Errors: Window}

\texttt{GetWindowAttributes} and \texttt{GetGeometry} are frequently used together in the X protocol. \texttt{LbxGetWinAttrAndGeom} allows the proxy to request the same information in one round trip.

The encoding for this request is on page 45.

\textbf{\texttt{LbxQueryFont}}

\[ \text{font: FONTABLE} \]

\[ \rightarrow \]

\[ \text{compression: BOOL} \]
\[ \text{tag: CARD32} \]
\[ \text{font-info: FONTINFO} /* optional */ \]
\[ \text{char-infos: LISTofCHARINFO or LISTofLBXCHARINFO} /* optional */ \]

where:

\[ \text{LBXCHARINFO: [left-side-bearing: INT6} \]
\[ \text{right-side-bearing: INT7} \]
\[ \text{character-width: INT6} \]
\[ \text{ascent: INT6} \]
\[ \text{descent: INT7]} \]
Errors: Font, Alloc

This request is used to replace the core QueryFont request and has identical semantics.

See “Tag Substitution in Requests” on page 4 for a description of the tag field and optional fields.

The compression field is True if the char-infos field is represented using LBXCHARINFO.

The per-character information will be encoded in an LBXCHARINFO when, for every character, the character-width, left-side-bearing, and ascent can each be represented in not more than 6 bits, and the right-side-bearing and descent can each be represented in not more than 7 bits, and the attributes field is identical the attributes field of the max_bounds of the font_info field of the font.

The encoding for this request is on page 39.

**LbxChangeProperty**

- window: WINDOW
- property: ATOM
- type: ATOM
- format: {0, 8, 16, 32}
- mode: [Replace, Prepend, Append]
- nUnits: CARD32

→

- tag: CARD32

This request is sent to the server when the client sends an X ChangeProperty request through the proxy. The size of the data is sent with this request, but not the property data itself. The server reply contains a tag identifier for the data, which is stored in the proxy. The proxy must not discard this data before it is sent to the server, or invalidated by the server. This means that before issuing an LbxStopProxy request, or exiting, the proxy must send LbxTagData requests for these items. If the server loses the connection before the information is sent back, the server should revert the property value to its last known value, if possible.

If the mode field is Prepend or Append, the tag refers only to the prepended or appended data.

If the tag in the reply is zero, then the change was ignored by the server, as defined in the security extension. The proxy should dump the associated data, since the server will never ask for it.

The encoding for this request is on page 40.

**LbxGetProperty**

- window: WINDOW
- property: ATOM
- type: ATOM or AnyPropertyType
- long-offset: CARD32
- long-length: CARD32
- delete: CARD8

→

- type: ATOM or None
- format: {0, 8, 16, 32}
- bytes-after: CARD32
Protocol

nItems: CARD32
  tag: CARD32
  value: LISTofINT8 or LISTofINT16 or LISTofINT32

This request may be used by the proxy as a substitution for a core SetProperty request. It allows tags to be used for property data that is unlikely to change often in value, but is likely to be fetched by multiple clients.

The LbxGetProperty request has the same arguments as the core SetProperty request. The reply for LbxGetProperty has all of the fields from the core SetProperty reply, but has the additional fields of nItems and tag.

In order to utilize tags in LbxGetProperty for a specific property the server must first send the complete property data to the proxy and associate this data with a tag. More precisely, the server sends an LbxGetProperty reply with a new tag, nItems set to the number of items in the property, the size of the property data in the reply length field, and the complete property data in value. The proxy stores the property data in its tag cache and associates it with the specified tag.

In response to future LbxGetProperty requests for the same property, if the server thinks that the proxy has the actual property data in its tag cache, it may choose to send an LbxGetProperty reply without the actual property data. In this case, the reply would include a non-zero tag, a zero reply length, and no data for value.

If the server chooses not to generate a tagged reply to LbxGetProperty, or for some reason is unable to do so, it would send a reply with a tag of zero, the size of the property data in the reply length field, and the complete property data in value.

The encoding for this request is on page 40.

**LbxPolyPoint**

  gc-and-drawable: LBXGCANDDRAWABLE
  points: LISTofLBXPOINT

Errors: Alloc and those given for the corresponding X request.

This request replaces the PolyPoint request. Not all PolyPoint requests can be represented as LbxPolyPoint requests.

The proxy will convert the representation of the points to be relative to the previous point, as described by previous coordinate mode in the X protocol.

The encoding for this request is on page 37.

**LbxPolyLine**

  gc-and-drawable: LBXGCANDDRAWABLE
  points: LISTofLBXPOINT

Errors: Alloc and those given for the corresponding X request.

This request replaces the PolyLine request. Not all PolyLine requests can be represented as LbxPolyline requests.
The proxy will convert the representation of the points to be relative to the previous point, as described by previous coordinate mode in the X protocol.

The encoding for this request is on page 37.

**LbxPolySegment**

\[ gc-and-drawable: \text{LBXGCANDDRAWABLE} \]
\[ segments: \text{LISTofLBXSEGMENT} \]

where:
\[ \text{LBXSEGEMENT: } [x1, y1, x2, y2: \text{LBXINT16}] \]

Errors: Alloc and those given for the corresponding X request.

This request replaces the *PolySegment* request. Not all *PolySegment* requests can be represented as *LbxPolySegment* requests.

For segments other than the first segment of the request, \([x1, y1]\) is relative to \([x1, y1]\) of the previous segment. For all segments, \([x2, y2]\) is relative to that segment’s \([x1, y1]\).

The encoding for this request is on page 37.

**LbxPolyRectangle**

\[ gc-and-drawable: \text{LBXGCANDDRAWABLE} \]
\[ rectangles: \text{LISTofLBXRECTANGLE} \]

Errors: Alloc and those given for the corresponding X request.

This request replaces the *PolyRectangle* request. Not all *PolyRectangle* requests can be represented as *LbxPolyRectangle* requests.

The encoding for this request is on page 38.

**LbxPolyArc**

\[ gc-and-drawable: \text{LBXGCANDDRAWABLE} \]
\[ arcs: \text{LISTofLBXARC} \]

Errors: Alloc and those given for the corresponding X request.

This request replaces the *PolyArc* request. Not all *PolyArc* requests can be represented as *LbxPolyArc* requests.

The encoding for this request is on page 38.

**LbxPolyFillRectangle**

\[ gc-and-drawable: \text{LBXGCANDDRAWABLE} \]
\[ rectangles: \text{LISTofLBXRECTANGLE} \]

Errors: Alloc and those given for the corresponding X request.
This request replaces the PolyFillRectangle request. Not all PolyFillRectangle requests can be represented as LbxPolyFillRectangle requests.

The encoding for this request is on page 38.

**LbxPolyFillArc**

- **gc-and-drawable**: LBXGCANDDRAWABLE
- **arcs**: LISTofLBXARC

Errors: Alloc and those given for the corresponding X request.

This request replaces the PolyFillArc request. Not all PolyFillArc requests can be represented as LbxPolyFillArc requests.

The encoding for this request is on page 39.

**LbxFillPoly**

- **gc-and-drawable**: LBXGCANDDRAWABLE
- **shape**: BYTE
- **points**: LISTofLBXPOINT

Errors: Alloc and those given for the corresponding X request.

This request replaces the FillPoly request. Not all FillPoly requests can be represented as LbxFillPoly requests.

The proxy will convert the representation of the points to be relative to the previous point, as described by previous coordinate mode in the X protocol.

The encoding for this request is on page 38.

**LbxCopyArea**

- **srcCache**: CARD8 /* source drawable */
- **gc-and-drawable**: LBXGCANDDRAWABLE
- **src-Drawable**: CARD32
- **src-x**: LBXPINT16
- **src-y**: LBXPINT16
- **width**: LBXCARD16
- **height**: LBXCARD16
- **dst-x**: LBXPINT16
- **dst-y**: LBXPINT16

Errors: Those given for the corresponding X request.

This request replaces the CopyArea request for requests within its encoding range.

The encoding for this request is on page 41.

**LbxCopyPlane**

- **bit-plane**: CARD32
LBX

src-cache: CARD8  /* cache reference for source drawable */
ge-and-drawable: LBXGCANDDRAWABLE
src-drawable: CARD32
src-x: LBXPINT16
src-y: LBXPINT16
width: LBXCARD16
height: LBXCARD16
dst-x: LBXPINT16
dst-y: LBXPINT16

Errors: Those given for the corresponding X request.

This request replaces the CopyPlane request for requests within its coding range.
The encoding for this request is on page 42.

LbxPolyText8

gc-and-drawable: LBXGCANDDRAWABLE
x: LBXPINT16
y: LBXPINT16
items: LISTofTEXTITEM8

Errors: Alloc, and those given for the corresponding X request.

This request replaces the PolyText8 request for requests within its encoding range.
The encoding for this request is on page 42.

LbxPolyText16

gc-and-drawable: LBXGCANDDRAWABLE
x: LBXPINT16
y: LBXPINT16
items: LISTofTEXTITEM16

Errors: Alloc, and those given for the corresponding X request.

This request replaces the PolyText16 request for requests within its encoding range.
The encoding for this request is on page 42.

LbxImageText8

gc-and-drawable: LBXGCANDDRAWABLE
nChars: CARD8
x: LBXPINT16
y: LBXPINT16
string: STRING8

Errors: Alloc, and those given for the corresponding X request.

This request replaces the ImageText8 request for requests within its encoding range.
Protocol

The encoding for this request is on page 42.

LbxImageText16

nChars: CARD8
gc-and-drawable: LBXGCANDDDRAWABLE
x: LBXPINT16
y: LBXPINT16
string: STRING16

Errors: Alloc, and those given for the corresponding X request.

This request replaces the ImageText16 request for requests within its encoding range.

The encoding for this request is on page 43.

LbxPutImage

compression-method: CARD8
format: {Bitmap, XYPixmap, ZPixmap} /* packed */
gc-and-drawable: LBXGCANDDDRAWABLE
width, height: LBXCARD16
dst-x, dst-y: LBXPINT16
depth: CARD8 /* packed */
left-pad: CARD8 /* packed */
pad-bytes: CARD8 /* packed */
data: LISTofBYTE

Errors: Alloc, Value

When the request can be usefully compressed, this request replaces the PutImage request. The compression-method parameter contains the opcode of a compression method returned in the LbxStartProxy reply. The pad-bytes parameter gives the number of unused pad bytes that follow the compressed image data. All other parameters are as in the X request. If the specified compression method is not recognized, the server returns a Value error.

The encoding for this request is on page 43.

LbxGetImage

drawable: DRAWABLE
x, y: INT16
width, height: CARD16
plane-mask: CARD32
format: {XYPixmap, ZPixmap}

→
depth: CARD8
x-length: CARD32
visual: VISUALID or None
compression-method: CARD8
data: LISTofBYTE

Errors: Alloc, Match, Value
This request can replace the GetImage request. The same semantics apply, with the following exceptions.

The *compression-method* field contains the opcode of the compression method used in the reply. The compression opcodes are supplied in the LbxStartProxy reply. The *x-length* field contains the length of the uncompressed version of the reply in 4 byte units.

A Value error is returned if the format is not recognized by the X server. A Match error is returned under the same circumstances as described by the GetImage request.

The encoding for this request is on page 44.

**LbxBeginLargeRequest**

\[ large-request-length: \text{CARD32} \]

**Errors:** Alloc

This request, along with the LbxLargeRequestData and LbxEndLargeRequest requests, is used to transport a large request in pieces. The smaller size of the resulting requests allows smoother multiplexing of clients on a single low bandwidth connection to the server. The resulting finer-grained multiplexing improves responsiveness for the other clients.

After a LbxBeginLargeRequest request is sent, multiple LbxLargeRequestData requests are sent to transport all of the data in the large request, and finally an LbxEndLargeRequest request is sent. The large-request-length field expresses the total length of the transported large request, expressed as the number of bytes in the transported request divided by four.

The encoding for this request is on page 44.

**LbxLargeRequestData**

\[ data: \text{LISTofBYTE} \]

**Errors:** Alloc

This request is used to carry the segments of a larger request, as described in the definition of LbxBeginLargeRequest. The data must be carried in order, starting with the request header, and each segment must be multiples of 4 bytes long. If the LbxLargeRequestData is not preceded by a corresponding LbxBeginLargeRequest, a BadAlloc error is generated.

The encoding for this request is on page 45.

**LbxEndLargeRequest**

**Errors:** Length, Alloc

As described in the definition of LbxBeginLargeRequest, LbxEndLargeRequest is used to signal the end of a series of LbxBeginLargeRequest requests. If the total length of the data transported by the LbxLargeRequestData requests does not match the large-request-length field of the preceding LbxBeginLargeRequest request, then a Length error occurs. If the LbxEndLargeRequest is not preceded by a corresponding LbxBeginLargeRequest, a BadAlloc error is generated. The request is executed in order for that client as if it were the request after the request preceding LbxEndLargeRequest.
Protocol

4.4 Events

**LbxSwitchEvent**

*client: CARD32*

Notify the proxy that the subsequent replies, events, and errors are relative to the specified client.

The encoding for this event is on page 48.

**LbxCloseEvent**

*client: CARD32*

Notify the proxy that the specified client’s connection to the server is closed.

The encoding for this event is on page 49.

**LbxInvalidateTagEvent**

*tag: CARD32*  
*tag-type: {Modmap, Keymap, Property, Font, ConnInfo}*

This message informs the proxy that the tag and the server data referenced by the tag are obsolete, and should be discarded. The tag type may be one of the following values: LbxTagTypeModmap, LbxTagTypeKeymap, LbxTagTypeProperty, LbxTagTypeFont, LbxTagTypeConnInfo.

The encoding for this event is on page 49.

**LbxSendTagDataEvent**

*tag: CARD32*  
*tag-type: {Property}*

The server sends this event to the proxy to request a copy of tagged data which is being stored by the proxy. The request contains a tag which was previously assigned to the data by the server. The proxy should respond to SendTagData by sending a TagData request to the server. The tag type may be one of the following values: LbxTagTypeProperty.

The encoding for this event is on page 49.

**LbxListenToOne**

*client: CARD32 or 0xffffffff*

When the server is grabbed, LbxListenToOne is sent to the proxy. As an X client, the proxy itself is unaffected by grabs, in order that it may respond to requests for data from the X server.

When the client grabbing the server is managed through the proxy, the proxy will permit messages from itself and the grabbing client to be sent immediately to the server, and may buffer requests from other clients of the proxy. The client is identified in the event.
When the client grabbing the server is not managed through the proxy, the client field in the event will be 0xffffffff. The proxy will communicate with the server, and it may buffer requests from other clients. The proxy will continue to handle new connections while the server is grabbed.

The server will send LbxListenToAll to the proxy when the server is ungrabbed. There is no time-out for this interval in the protocol.

The encoding for this event is on page 49.

**LbxListenToAll**

Notify the proxy that the server has been ungrabbed, and that the proxy may now send all buffered client requests on to the server.

The encoding for this event is on page 49.

**LbxQuickMotionDeltaEvent**

\[
\begin{align*}
\delta t & : \text{CARD8} \\
\delta x & : \text{INT8} \\
\delta y & : \text{INT8}
\end{align*}
\]

This event is used as a replacement for the MotionNotify event when possible. The fields are used as deltas to the most recent MotionNotify event encoded as a MotionNotify event, LbxQuickMotionDeltaEvent, or LbxMotionDeltaEvent. Not every MotionNotify event can be encoded as a LbxQuickMotionDeltaEvent.

The encoding for this event is on page 50.

**LbxMotionDeltaEvent**

\[
\begin{align*}
\delta x & : \text{INT8} \\
\delta y & : \text{INT8} \\
\delta t & : \text{CARD16} \\
\delta s & : \text{CARD16}
\end{align*}
\]

This event is used as a replacement for the MotionNotify event when possible. The fields are used as deltas to the most recent MotionNotify event encoded as a MotionNotify event, LbxQuickMotionDeltaEvent, or LbxMotionDeltaEvent. Not every MotionNotify event can be encoded as a LbxMotionDeltaEvent.

The encoding for this event is on page 50.

**LbxReleaseCmapEvent**

\[
\begin{align*}
\text{cm} & : \text{Colormap}
\end{align*}
\]

This event notifies the proxy that it must release the grab on this colormap via the ReleaseCmap request. See “LbxReleaseCmap” on page 19.

The encoding for this event is on page 50.

**LbxFreeCellsEvent**
Algorithm Naming

colormap: Colormap
pixelStart, pixelEnd: CARD32

The LbxFreeCells event is sent to a proxy that has a colormap grabbed to notify the proxy that the reference count of the described cells were decremented to zero by the server or another proxy. The reference count includes those by this proxy. The proxy must update its copy of the colormap state accordingly if the colormap is still grabbed, or if the proxy may in the future grab the colormap using smart-grab mode. See “LbxGrabCmap” on page 17.

The pixelStart and pixelEnd fields of the event denote a continuous range of cells that were freed.

The encoding for this event is on page 50.

4.5 Responses

Responses are messages from the server to the proxy that not, strictly speaking, events, replies or errors.

LbxDeltaResponse

  count: CARD8
  cache-index: CARD8
  diffs: LISTofDIFFITEM

This response carries an event, reply, or error that has been encoded relative to a message in the response delta cache. The cache-index field is the index into the cache. Each entry in diffs provides a byte offset and replacement value to use in reconstructing the response.

The encoding for this event is on page 51.

5 Algorithm Naming

To avoid potential clashes between different but similar algorithms for stream, bitmap, and pixmap compression, the following naming scheme will be adhered to:

Each algorithm has a unique name, which is a STRING8, of the following form:

<organization>-<some-descriptive-name>

The organization field above is the organization name as registered in section 1 of the X Registry (the registry is provided as a free service by the X Consortium.) This prevents conflicts among different vendor’s extensions.

As an example, the X Consortium defines a zlib-based stream compression algorithm called XC-ZLIB.

6 Encoding

The syntax and types used in the encoding are taken from the X protocol encoding. Where LBX defines new types, they are defined earlier in this document.

As in the X protocol, in various cases, the number of bytes occupied by a component will be specified by a lowercase single-letter variable name instead of a specific numeric value, and often some other component
will have its value specified as a simple numeric expression involving these variables. Components specified with such expressions are always interpreted as unsigned integers. The scope of such variables is always just the enclosing request, reply, error, event, or compound type structure.

For unused bytes, the encode-form is:

\[ N \text{ unused} \]

If the number of unused bytes is variable, the encode-form typically is:

\[ p \text{ unused, } p=\text{pad}(E) \]

where \( E \) is some expression, and \( \text{pad}(E) \) is the number of bytes needed to round \( E \) up to a multiple of four.

\[ \text{pad}(E) = (4 - (E \mod 4)) \mod 4 \]

In many of the encodings, the length depends on many variable length fields. The variable \( L \) is used to indicate the number of padded 4 byte units needed to carry the request. Similarly, the variable \( L_{\text{pad}} \) indicates the number of bytes needed to pad the request to a 4 byte boundary.

For counted lists there is a common encoding of \( \text{NLISTofFOO} \):

\[ \text{NLISTofFOO} \]

\[ 1 \text{ m} \text{ num items} \]
\[ \text{m LISTofFOO items} \]

For cached GC and Drawables:

\[ \text{LBXGCANDDRAWUPDATE} \]

\[ 4 \text{ or } 0 \text{ DRAWBLE} \text{ optional drawable} \]
\[ 4 \text{ or } 0 \text{ GC} \text{ optional GC} \]

\[ \text{LBXGCANDDRAWABLE} \]

\[ 8 \text{ LBXGCANDDRAWENT cache-entries} \]
\[ 8 \text{ unused} \]
\[ \text{m LBXGCANDDRAWUPDATE optional GC and Drawable} \]

### 6.1 Errors

**LbxClient**

\[ 1 \text{ 0 Error} \]
\[ 1 \text{ CARD8 error-base + 0} \]
\[ 2 \text{ CARD16 sequence number} \]
\[ 4 \text{ unused} \]
\[ 2 \text{ CARD16 lbx opcode} \]
\[ 1 \text{ CARD8 major opcode} \]
\[ 21 \text{ unused} \]

### 6.2 Requests

**LbxQueryVersion**
Encoding

1 CARD8 opcode
1 0 lbx opcode
2 1 request length

→

1 1 Reply
1 unused
2 CARD16 sequence number
4 0 reply length
2 CARD16 major version
2 CARD16 minor version
20 unused

The description of this request is on page 10.

LbxStartProxy

1 CARD8 opcode
1 1 lbx opcode
2 L request length
n NLISTofOPTION-REQUEST options
p unused, p=pad(n)

OPTION-REQUEST

1 OPTCODE option-code
m OPTLEN option-request-byte-length, (b=m+a+1)
a DELTAOPT or option
NLISTofNAMEDOPT or
NLISTofSTR or
NLISTofPIXMAPMETHOD or
BOOL

The encoding of the option field depends on the option-code. See Table 1, “StartProxy Options,” on page 12.

1 OPTCODE option-code
0 LbxOptionDeltaProxy
1 LbxOptionDeltaServer
2 LbxOptionStreamCompression
3 LbxOptionBitmapCompression
4 LbxOptionPixmapCompression
5 LbxOptionMessageCompression /* also known as squishing */
6 LbxOptionUseTags
7 LbxOptionColormapAllocation
255 LbxOptionExtension

OPTLEN has two possible encodings, depending on the size of the value carried:

OPTLEN
1 CARD8 b (0 < b <= 255)

OPTLEN
1 0 long length header
1 c length0, c = b >> 8
1 d length1, d= b & #xff
DELTAOPT

1 CARD8 min-cache-size
1 CARD8 max-cache-size
1 CARD8 preferred-cache-size
1 CARD8 min-message-length
1 CARD8 max-message-length (in 4-byte units)
1 CARD8 preferred-message-length

NAMEDOPT

f STR type-name
1 g+1 option-data-length
1 LISTofBYTE option-data (option specific)

PIXMAPMETHOD

h STR name
1 BITMASK format mask
1 j depth count
1 LISTofCARD8 depths

→

1 1 Reply
1 CARD8 count
0xff options in request cannot be decoded
2 CARD16 sequence number
4 (a+p-32)/4 reply length
a LISTofCHOICE options-reply
p unused, if (n<24) p=24-n else p=pad(n)

CHOICE

1 CARD8 request-option-index
b OPTLEN reply-option-byte-length
c DELTACHoice or INDEXEDChoice or
   NLISTofINDEXEDOPT or
   NLISTofPIXMAPCHOICE or
   BOOL or
   INDEXEDChoice

The encoding of the choice field depends on the option-code. See Table 1, "StartProxy Options," on page 12.

DELTACHoice

1 CARD8 preferred cache size
1 CARD8 preferred message length in 4-byte units

INDEXEDChoice

1 CARD8 index
1 LISTofBYTE data
Encoding

PIXMAPCHOICE

1 CARD8 index
1 CARD8 opcode
1 BITMASK format mask
e NLISTofCARD8 depths

The description of this request is on page 11.

LbxStopProxy

1 CARD8 opcode
1 2 lbx opcode
2 1 request length

The description of this request is on page 14.

LbxSwitch

1 CARD8 opcode
1 3 lbx opcode
2 2 request length
4 CARD32 client

The description of this request is on page 15.

LbxNewClient

1 CARD8 opcode
1 4 lbx opcode
2 L request length
4 CARD32 client

The remaining bytes of the request are the core connection setup.

If the connection is rejected, a core connection reply is sent. Otherwise the reply has the form:

1 BOOL success
1 change type
0 no-deltas
1 normal-client-deltas
2 app-group-deltas
2 CARD16 major version
2 CARD16 minor version
2 1 + a length
4 CARD32 tag id

The remaining bytes depend on the value of change-type and length.

For no-deltas, the remaining bytes are the "additional data" bytes of the core reply. (a = length of core reply, in 4 byte quantities).

For normal-client-deltas, the additional bytes have the form, with a length (a = 1 +b):

4 CARD32 resource id base
4b LISTofSETofEVENT root input masks
For app-group-deltas, the additional bytes have the following form, with a length of \((a = 1 + 4c)\):

- 4 CARD32 resource id base
- 4 WINDOW root id base
- 4 VISUALID visual
- 4 COLORMAP colormap
- 4 CARD32 white pixel
- 4 CARD32 black pixel
- 4c LISTofSETofEVENT root input masks

The description of this request is on page 14.

**LbxCloseClient**

1 CARD8 opcode
1  5 lbx opcode
2  2 request length
4  CARD32 client

The description of this request is on page 15.

**LbxModifySequence**

1 CARD8 opcode
1  6 lbx opcode
2  2 request length
4  CARD32 offset to sequence number

The description of this request is on page 16.

**LbxAllowMotion**

1 CARD8 opcode
1  7 lbx opcode
2  2 request length
4  CARD32 number of MotionNotify events

The description of this request is on page 16.

**LbxIncrementPixel**

1 CARD8 opcode
1  8 lbx opcode
2  3 request length
4  COLORMAP colormap
4  CARD32 pixel

The description of this request is on page 20.

**LbxDelta**

1 CARD8 opcode
1  9 lbx opcode
2 1+(2n +p+2)/4 request length
1  n count of diffs
Encoding

1 CARD8                     cache index
2n LISTofDIFFITEM           offsets and differences
p unused, p=pad(2n + 2)

The description of this request is on page 20.

**LbxGetModifierMapping**

1 CARD8                     opcode
1 10 lbx opcode
2 1 request length

→

1 1 Reply
1 n keycodes-per-modifier
2 CARD16 sequence number
4 2n reply length
4 CARD32 tag
20 unused
8n LISTofKEYCODE keycodes

The description of this request is on page 20.

**LbxInvalidateTag**

1 CARD8                     opcode
1 12 lbx opcode
2 2 request length
4 CARD32 tag

The description of this request is on page 16.

**LbxPolyPoint**

1 CARD8                     opcode
1 13 lbx opcode
2 1+(m+n+p)/4 request length
m LBXGCANDDRAWABLE cache entries
n LISTofLBXPOINT points (n is data-dependent)
p 0 unused, p=Lpad

The description of this request is on page 23.

**LbxPolyLine**

1 CARD8                     opcode
1 14 lbx opcode
2 1+(m+n+p)/4 request length
m LBXGCANDDRAWABLE cache entries
n LISTofLBXPOINT points (n is data-dependent)
p 0 unused, p=Lpad

The description of this request is on page 23.

**LbxPolySegment**
LbxPolyRectangle

1 CARD8
1 16
2 1+(m+n+p)/4
m LBXGCANDDRAWABLE
n LISTofLBXRECTANGLE
p 0

The description of this request is on page 24.

LbxPolyArc

1 CARD8
1 17
2 1+(m+n+p)/4
m LBXGCANDDRAWABLE
n LISTofLBXARCS
p 0

The description of this request is on page 24.

LbxFillPoly

1 CARD8
1 18
2 1+(3+m+n+p)/4
m LBXGCANDDRAWUPDATE
l 0 Complex
1 1 Nonconvex
2 2 Convex
1 p pad byte count
m LBXGCANDDRAWUPDATE
n LISTofLBXPOINT
p 0

The description of this request is on page 25.

LbxPolyFillRectangle

1 CARD8
1 19
2 1+(m+n+p)/4
m LBXGCANDDRAWABLE

The description of this request is on page 24.
n  LISTofLBXRECTANGLE  rectangles (n is data-dependent)
p  0  unused, p=Lpad

The description of this request is on page 24.

**LbxPolyFillArc**

1  CARD8  opcode
1  20  lbx opcode
2  1+(m+n+p)/4  request length
m  LBXGCANDDRAWABLE  cache entries
n  LISTofLBXARC  arcs (n is data-dependent)
p  0  unused, p=Lpad

The description of this request is on page 25.

**LbxGetKeyboardMapping**

1  CARD8  opcode
1  21  lbx opcode
2  2  request length
1  KEYCODE  fi rst keycode
1  m  count
2  unused
→
1  1  Reply
1  n  keysyms-per-keycode
2  CARD16  sequence number
4  nm  reply length (m = count field from the request)
4  CARD32  tag
20  unused
4nm  LISTofKEYSYM  keysyms

The description of this request is on page 20.

**LbxQueryFont**

1  CARD8  opcode
1  22  lbx opcode
2  2  request length
4  FONTABLE  font
→
1  1  Reply
1  BOOL  compression
2  CARD16  sequence number
4  L  reply length
4  CARD32  tag
20  unused
All of the following is conditional:
12  CHARINFO  min-bounds
4  unused
12  CHARINFO  max-bounds
4  unused
2  CARD16  min-char-or-byte2
LbxChangeProperty

```c
1 CARD8  opcode
2 23     lbx opcode
2 6      request length
4 WINDOW window
4 ATOM property
4 ATOM type
1 CARD8 format
1
0  Replace
1  Preprend
2  Append
2
CARD32 length of data in format units
(= n for format = 8)
(= n/2 for format = 16)
(= n/4 for format = 32)
```

The description of this request is on page 22.

LbxGetProperty

```c
1 CARD8  opcode
1 24     lbx opcode
2 7      request length
4 WINDOW window
```

The description of this request is on page 22.
Encoding

4  ATOM  property
4  ATOM  type
0  AnyPropertyType
1  CARD8  delete
3  unused
4  CARD32  long-offset
4  CARD32  long-length
→
1  1  Reply
1  CARD8  format
2  CARD16  sequence number
4  CARD32  reply length
4  ATOM  type
0  None  bytes-after
4  CARD32  length of value in format units
(= 0 for format = 0)
(= n for format = 8)
(= n/2 for format = 16)
(= n/4 for format = 32)

4  CARD32  tag
8  unused

The description of this request is on page 22.

LbxTagData

1  CARD8  opcode
1  25  lbx opcode
2  3+(n+p)/4  request length
4  CARD32  tag
4  CARD32  length of data in bytes
n  LISTofBYTE  data
p  unused, p=pad(n)

The description of this request is on page 16.

LbxCopyArea

1  CARD8  opcode
1  26  lbx opcode
2  L  request length
1  CARD8  source drawable cache entry
1  LBXGCANDDRAWENT  cache entries
4 or 0  DRAWARE  optional source drawable
b  LBXGCANDDRAWUPDATE  optional gc and dest drawable
c  LBXPINT16  src-x
d  LBXPINT16  src-y
e  LBXPINT16  dst-x
f  LBXPINT16  dst-y
g  LBXCARD16  width
h  LBXCARD16  height
p  unused, p=Lpad
The description of this request is on page 25.

**LbxCopyPlane**

1  CARD8  opcode
1  27  lbx opcode
2  L  request length
4  CARD32  bit plane
1  CARD8  source drawable cache entry
1  LBXGCANDDRAWENT  cache entries
4 or 0  DRAWARABLE  optional source drawable
b  LBXGCANDDRAWUPDATE  optional gc and dest drawable
c  LBPXINT16  src-x
d  LBPXINT16  src-y
e  LBPXINT16  dst-x
f  LBPXINT16  dst-y
g  LBXCARD16  width
h  LBXCARD16  height
p  unused, p=Lpad

The description of this request is on page 25.

**LbxPolyText8**

1  CARD8  opcode
1  28  lbx opcode
2  L  request length
1  LBXGCANDDRAWENT  cache entries
a  LBXGCANDDRAWUPDATE  optional gc and drawable
b  LBPXINT16  x
c  LBPXINT16  y
n  LISTofTEXTITEM8  items
p  unused, p=Lpad

The description of this request is on page 26.

**LbxPolyText16**

1  CARD8  opcode
1  29  lbx opcode
2  L  request length
1  LBXGCANDDRAWENT  cache entries
a  LBXGCANDDRAWUPDATE  optional gc and drawable
b  LBPXINT16  x
c  LBPXINT16  y
2n  LISTofTEXTITEM16  items
p  unused, p=Lpad

The description of this request is on page 26.

**LbxImageText8**

1  CARD8  opcode
1  30  lbx opcode
2  L
1  LBXGCANDDRAWENT
  request length
  cache entries
  optional gc and drawable
a  LBXGCANDDRAWUPDATE
b  LBXPINT16
c  LBXPINT16
n  STRING8
p  unused, p=Lpad

The description of this request is on page 26.

LbxImageText16
1  CARD8
1  31
2  L
1  LBXGCANDDRAWENT
  request length
  cache entries
  optional gc and drawable
a  LBXGCANDDRAWUPDATE
b  LBXPINT16
c  LBXPINT16
2n STRING16
p  unused, p=Lpad

The description of this request is on page 27.

LbxQueryExtension
1  CARD8
1  32
2  2+(n+p)/4
4  n
n  STRING8
p  unused, p=pad(n)

1  1
n  STRING8
p  unused, p=pad(n)

1  CARD16
2  0 or 2*(m + p)
1  BOOL
1  CARD8
1  CARD8
1  CARD8
20  unused
m  LISTofMASK
p  unused, p=pad(m)
m  LISTofMASK
p  unused, p=pad(m)

The description of this request is on page 11.

LbxPutImage
1  CARD8
1  33
2  L
  request length
  lbx opcode
  optional reply-mask
  optional event-mask
  unused, p=pad(m)
  unused, p=pad(m)
  unused
  optional reply-mask
  optional event-mask
  unused, p=pad(m)
  unused
  major opcode
  fi rst error
  fi rst event
  present
  reply length, m = (n+7)/8
  sequence number
  number of requests in the extension
  extension name
  extension name
  major opcode
  fi rst event
  fi rst error
  unused
  unused, p=pad(m)
  unused, p=pad(m)
  unused, p=pad(m)
  unused, p=pad(m)

The description of this request is on page 11.
LBX

1  CARD8                      compression method
1  LBXGCANDDRAWEVENT         cache entries
a  PIPACKED                  bit-packed
b  LBXGCANDDRAWUPDATE       optional gc and drawable
c  LBXCARD16                 width
d  LBXCARD16                 height
e  LBXPINT16                 x
f  LBXPINT16                 y
n  LISTofBYTE               compressed image data
p                         unused, p=Lpad

If there is no left padding and the depth is less than or equal to nine, PIPACKED is encoded as follows:

PIPACKED
  1  #x80 | (format << 5) | ((depth -1) << 2)

Otherwise PIPACKED is defined as:

PIPACKED
  1  (depth -1) << 2)
  1  (format << 5) | left-pad

The description of this request is on page 27.

LbxGetImage

1  CARD8          opcode
1  34             lbx opcode
2  6              request length
4  DRAWABLE       drawable
2  INT16          x
2  INT16          y
2  CARD16         width
2  CARD16         height
4  CARD32         plane mask
1  CARD8          format
3              unused

1  1               Reply
1  CARD8          depth
2  CARD16         sequence number
4  (n+p)/4        reply length
4  (m+p)/4        X reply length; if uncompressed, m=n
4  VISUALID       visual
  0    None
1              compression method
15             unused
n  LISTofBYTE     data
p             unused, p=pad(n)

The description of this request is on page 27.

LbxBeginLargeRequest

1  CARD8          opcode
LbxLargeRequestData

1 CARD8
1 36 lbx opcode
2 1+n request length
4n LISTofBYTE

The description of this request is on page 28.

LbxEndLargeRequest

1 CARD8
1 37 lbx opcode
2 1 request length

The description of this request is on page 28.

LbxInternAtoms

1 CARD8
1 38 lbx opcode
2 1+(2+m+n+p)/4 request length
2 m num-atoms
n LISTofLONGSTR
p pad p=Lpad
→
1 1 Reply
1 unused
2 CARD16 sequence number
4 a reply length, a = MAX(m - 6, 0)
4*m LISTofATOM atoms
p pad p = MAX(0, 4*(6 - m))
LONGSTR
2 c string length
0 c STRING8 string

The description of this request is on page 19.

LbxGetWinAttrAndGeom

1 CARD8
1 39 lbx opcode
2 2 request length
4 CARD32 window id
→
1 1 Reply
The description of this request is on page 21.

LbxGrabCmap

If smart-grab is true, the reply is as follows:

If smart-grab is false, the reply is as follows:
Encoding

1 1 Reply
1
   #x40 auto-release
   #x20 three-channels
   #x10 two-byte-pixels
   lower four bits specifies bits-per-pixel
2 CARD16 sequence number
4 L reply length
m CHAN or CHANNELS cells (CHAN if !three-channels)
p 0 pad(m)

CHANNELS
a CHAN red
l 5 next channel
b CHAN green
l 5 next channel
c CHAN blue
l 0 list end

CHAN
d LISTofLBXPIXEL

LBXPIXEL
e PIXELPRIVATE or
   PIXELPRIVATERANGE or
   PIXELALLOC or
   PIXELALLOCRANGE

PIXELPRIVATE
l 1 pixel-private
f PIXEL pixel

PIXEL
f CARD8 or CARD16 (CARD8 if !two-byte-pixels)

PIXELPRIVATERANGE
l 2 pixel-private-range
f PIXEL fi st-pixel
f PIXEL last-pixel

PIXELALLOC
l 3 pixel-private
f PIXEL pixel
g COLORSINGLE or COLORTRIPLE color (COLORSINGLE if three-channels)

COLORSINGLE
h CARD8 or CARD16 value (CARD8 if bits-per-rgb <= 7)

COLORTRIPLE
h COLORSINGLE red
h COLORSINGLE green
h COLORSINGLE blue
PIXELALLOCARGE
l 4 pixel-private
f PIXEL fi rst-pixel
f PIXEL last-pixel
j LISTofCOLORSINGLE or color (COLORSINGLE if three-channels)
LISTofCOLORTRIPLE

The description of this request is on page 17.

LbxReleaseCmap

1 CARD8 opcode
1 41 lbx opcode
2 2 request length
4 COLORMAP cmap

The description of this request is on page 19.

LbxAllocColor

1 CARD8 opcode
1 42 lbx opcode
2 5 request length
4 COLORMAP colormap
4 CARD32 pixel
2 CARD16 red
2 CARD16 green
2 CARD16 blue
2 unused

The description of this request is on page 19.

LbxSync

1 CARD8 opcode
1 43 lbx opcode
2 1 request length
→
1 1 Reply
1 n unused
2 CARD16 sequence number
4 0 reply length
24 unused

The description of this request is on page 16.

6.3 Events

LbxSwitchEvent

1 base + 0 code
1 0 lbx type
2 CARD16 sequence number
Encoding

4  CARD32  client
24  unused

The description of this event is on page 29.

LbxCloseEvent

1  base + 0  code
1  1  lbx type
2  CARD16  sequence number
4  CARD32  client
24  unused

The description of this event is on page 29.

LbxInvalidateTagEvent

1  base + 0  code
1  3  lbx type
2  CARD16  sequence number
4  CARD32  tag
4  tag-type
1  LbxTagTypeModmap
2  LbxTagTypeKeymap
3  LbxTagTypeProperty
4  LbxTagTypeFont
5  LbxTagTypeConnInfo
20  unused

The description of this event is on page 29.

LbxSendTagDataEvent

1  base + 0  code
1  4  lbx type
2  CARD16  sequence number
4  CARD32  tag
4  tag-type
3  LbxTagTypeProperty
20  unused

The description of this event is on page 29.

LbxListenToOne

1  base + 0  code
1  5  lbx type
2  CARD16  sequence number
4  CARD32  client
#xFFFFFFFF a client not managed by the proxy
24  unused

The description of this event is on page 29.
LbxListenToAll

1 base + 0 code
1 6 lbx type
2 CARD16 sequence number
28 unused

The description of this event is on page 30.

LbxQuickMotionDeltaEvent

1 base + 1 code
1 CARD8 delta-time
1 INT8 delta-x
1 INT8 delta-y

This event is not padded to 32 bytes.

The description of this event is on page 30.

LbxMotionDeltaEvent

1 base + 0 code
1 7 lbx type
1 INT8 delta-x
1 INT8 delta-y
2 CARD16 delta-time
2 CARD16 delta-sequence

This event is not padded to 32 bytes.

The description of this event is on page 30.

LbxReleaseCmapEvent

1 base + 0 code
1 8 lbx type
2 CARD16 sequence number
4 COLORMAP colormap
24 unused

The description of this event is on page 30.

LbxFreeCellsEvent

1 base + 0 code
1 9 lbx type
2 CARD16 sequence number
4 COLORMAP colormap
4 PIXEL pixel start
4 PIXEL pixel end
16 unused

The description of this event is on page 30.
6.4 Re-encoding of X Events

The X protocol requires all X events to be 32 bytes. The LBX server reduces the number of bytes sent between the server and the proxy for some X events by not appending unused pad bytes to the event data. The offsets of X event data are unchanged. The proxy will pad the events to 32 bytes before passing them on to the client.

LBX reencodes X event representations into the following sizes, if squishing is enabled:

- KeyOrButton 32
- EnterOrLeave 32
- Keymap 32
- Expose 20
- GraphicsExposure 24
- NoExposure 12
- VisibilityNotify 12
- CreateNotify 24
- DestroyNotify 12
- UnmapNotify 16
- MapNotify 16
- MapRequest 12
- Reparent 24
- ConfigureNotify 28
- ConfigureRequest 28
- GravityNotify 16
- ResizeRequest 12
- Circulate 20
- Property Notify 20
- SelectionClear 20
- SelectionRequest 28
- SelectionNotify 24
- ColormapNotify 16
- MappingNotify 8
- ClientMessage 32
- Unknown 32

6.5 Responses

LbxDeltaResponse

- 1 event_base + 0 event code
- 1 2 lbx type
- 2 1+(2+2n+p)/4 request length
- 1 n count of diffs
- 1 CARD8 cache index
- 2n LISTofDIFFITEM offsets and differences
- p unused, p=pad(2n)

The description of this response is on page 31.
# Index

## Introduction

- Data Flow ................................................................. 3
- Tags ............................................................................. 3
  - Tag Substitution in Requests ..................................... 4
  - Property Tags ............................................................ 4

## Short-circuiting ......................................................... 4

## Graphics Re-encoding ............................................... 5

## Motion events ............................................................ 5

## Event Squishing .......................................................... 5

## Master Client ............................................................... 6

## Multiplexing of Clients .................................................. 6

## Swapping ..................................................................... 6

## Delta cache ................................................................. 6

## Stream Compression ...................................................... 7

## Authentication Protocols ................................................ 7

## C Library Interfaces ...................................................... 7

### Application Library Interfaces ........................................... 7
- XLbxQueryVersion ..................................................... 7

### Proxy Library Interfaces .................................................. 8
- XLbxQueryExtension ................................................... 8
- XLbxGetEventBase ..................................................... 8

## Protocol ....................................................................... 8

### Syntactic Conventions and Common Types .......................... 8

### Errors ....................................................................... 10

### Requests ..................................................................... 10
- Requests Initiated by the Proxy or by the Client .................. 10
  - LbxQueryVersion ..................................................... 10
- Requests Initiated or Substituted by the Proxy ...................... 11
  - LbxQueryExtension ................................................... 11
- Control Requests Initiated by the Proxy ............................... 11
  - LbxStartProxy ......................................................... 11
  - LbxStopProxy .......................................................... 14
  - LbxNewClient .......................................................... 14
  - LbxCloseClient ....................................................... 15
  - LbxSwitch ............................................................... 15
  - LbxSync ................................................................. 16
  - LbxModifySequence .................................................. 16
  - LbxAllowMotion ....................................................... 16
  - LbxInvalidateTag ...................................................... 16
  - LbxTagData .............................................................. 16
  - LbxGrabCmap ........................................................... 17
  - LbxReleaseCmap ...................................................... 19
  - LbxInternAtoms ......................................................... 19
- Substitution Requests ................................................... 19
  - LbxAllocColor ........................................................ 19
  - LbxIncrementPixel ................................................... 20
  - LbxDelta ................................................................. 20
  - LbxGetModifierMapping .............................................. 20
  - LbxGetKeyboardMapping ............................................. 20
LbxInvalidateTag ................................................................. 37
LbxPolyPoint ........................................................................ 37
LbxPolyLine ........................................................................... 37
LbxPolySegment ..................................................................... 37
LbxPolyRectangle ................................................................... 38
LbxPolyArc ................................................................................ 38
LbxFillPoly ............................................................................... 38
LbxPolyFillRectangle .............................................................. 38
LbxPolyFillArc .......................................................................... 39
LbxGetKeyboardMapping ......................................................... 39
LbxQueryFont .......................................................................... 39
LbxChangeEvent ........................................................................ 40
LbxGetProperty ........................................................................ 40
LbxTagData ............................................................................... 41
LbxCopyArea ............................................................................ 41
LbxCopyPlane ........................................................................... 41
LbxCopyPlane ........................................................................... 42
LbxPolyText8 ............................................................................ 42
LbxPolyText16 ......................................................................... 42
LbxImageText8 ......................................................................... 42
LbxImageText16 ....................................................................... 43
LbxQueryExtension ................................................................. 43
LbxPutImage .............................................................................. 43
LbxGetImage ............................................................................. 44
LbxBeginLargeRequest ............................................................ 44
LbxLargeRequestData ............................................................. 44
LbxEndLargeRequest ............................................................... 44
LbxInternAtoms ........................................................................ 45
LbxGetWinAttrAndGeom ......................................................... 45
LbxGrabCmap ........................................................................... 46
LbxReleaseCmap ...................................................................... 48
LbxAllocColor ......................................................................... 48
LbxSync .................................................................................... 48

6.3 Events ............................................................................ 48
LbxSwitchEvent ........................................................................ 48
LbxCloseEvent ......................................................................... 49
LbxInvalidateTagEvent ............................................................ 49
LbxSendTagDataEvent ............................................................. 49
LbxListenToOne ......................................................................... 49
LbxListenToAll .......................................................................... 50
LbxQuickMotionDeltaEvent ..................................................... 50
LbxMotionDeltaEvent ............................................................... 50
LbxReleaseCmapEvent ............................................................. 50
LbxFreeCellsEvent ................................................................. 50

6.4 Re-encoding of X Events ....................................................... 51
6.5 Responses ......................................................................... 51
LbxDeltaResponse ..................................................................... 51