



```
    Display *dpy,  
    Window window,  
    XmbufWindowAttributes *attributes);  
  
void XmbufChangeWindowAttributes(  
    Display *dpy,  
    Window window,  
    unsigned long valuemask,  
    XmbufSetWindowAttributes *attributes);  
  
Status XmbufGetBufferAttributes(  
    Display *dpy,  
    Multibuffer buffer,  
    XmbufBufferAttributes *attributes);  
  
void XmbufChangeBufferAttributes(  
    Display *dpy,  
    Multibuffer buffer,  
    unsigned long valuemask,  
    XmbufSetBufferAttributes *attributes);  
  
Status XmbufGetScreenInfo(  
    Display *dpy,  
    Drawable drawable,  
    int *nmono_return,  
    XmbufBufferInfo **mono_info_return,  
    int *nstereo_return,  
    XmbufBufferInfo **stereo_info_return);  
  
Window XmbufCreateStereoWindow(  
    Display *dpy,  
    Window parent,  
    int x,  
    int y,  
    unsigned int width,  
    unsigned int height,  
    unsigned int border_width,  
    int depth,  
    unsigned int class,          /* InputOutput, InputOnly*/  
    Visual *visual,
```

```

    unsigned long valuemask,
    XSetWindowAttributes *attributes,
    Multibuffer *left_return,
    Multibuffer *right_return);

```

## STRUCTURES

*Events:*

```

typedef struct {
    int type; /* of event */
    unsigned long serial; /* # of last request processed by server */
    int send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Multibuffer buffer; /* buffer of event */
    int state; /* see Clobbered constants above */
} XmbufClobberNotifyEvent;

```

```

typedef struct {
    int type; /* of event */
    unsigned long serial; /* # of last request processed by server */
    int send_event; /* true if this came from a SendEvent request */
    Display *display; /* Display the event was read from */
    Multibuffer buffer; /* buffer of event */
} XmbufUpdateNotifyEvent;

```

*Per-window attributes that can be got:*

```

typedef struct {
    int displayed_index; /* which buffer is being displayed */
    int update_action; /* Undefined, Background, Untouched, Copied */
    int update_hint; /* Frequent, Intermittent, Static */
    int window_mode; /* Mono, Stereo */
    int nbuffers; /* Number of buffers */
    Multibuffer *buffers; /* Buffers */
} XmbufWindowAttributes;

```

*Per-window attributes that can be set:*

```

typedef struct {
    int update_hint; /* Frequent, Intermittent, Static */
} XmbufSetWindowAttributes;

```

*Per-buffer attributes that can be got:*

```
typedef struct {
    Window window;    /* which window this belongs to */
    unsigned long event_mask; /* events that have been selected */
    int buffer_index; /* which buffer is this */
    int side;         /* Mono, Left, Right */
} XmbufBufferAttributes;
```

*Per-buffer attributes that can be set:*

```
typedef struct {
    unsigned long event_mask; /* events that have been selected */
} XmbufSetBufferAttributes;
```

*Per-screen buffer info (there will be lists of them):*

```
typedef struct {
    VisualID visualid; /* visual usable at this depth */
    int max_buffers; /* most buffers for this visual */
    int depth; /* depth of buffers to be created */
} XmbufBufferInfo;
```

## DESCRIPTION

The application programming library for the *X11 Double-Buffering, Multi-Buffering, and Stereo Extension* contains the interfaces described below. With the exception of *XmbufQueryExtension*, if any of these routines are called with a display that does not support the extension, the *ExtensionErrorHandler* (which can be set with *XSetExtensionErrorHandler* and functions the same way as *XSetErrorHandler*) will be called and the function will then return.

*XmbufQueryExtension* returns *True* if the multibuffering/stereo extension is available on the given display. If the extension exists, the value of the first event code (which should be added to the event type constants *MultibufferClobberNotify* and *MultibufferUpdateNotify* to get the actual values) is stored into *event\_base\_return* and the value of the first error code (which should be added to the error type constant *MultibufferBadBuffer* to get the actual value) is stored into *error\_base\_return*.

*XmbufGetVersion* gets the major and minor version numbers of the extension. The return value is zero if an error occurs or non-zero if no error happens.

*XmbufCreateBuffers* requests that "count" buffers be created with the given *update\_action* and

update\_hint and be associated with the indicated window. The number of buffers created is returned (zero if an error occurred) and buffers\_update is filled in with that many Multibuffer identifiers.

*XmbufDestroyBuffers* destroys the buffers associated with the given window.

*XmbufDisplayBuffers* displays the indicated buffers their appropriate windows within max\_delay milliseconds after min\_delay milliseconds have passed. No two buffers may be associated with the same window or else a Match error is generated.

*XmbufGetWindowAttributes* gets the multibuffering attributes that apply to all buffers associated with the given window. The list of buffers returns may be freed with *XFree*. Returns non-zero on success and zero if an error occurs.

*XmbufChangeWindowAttributes* sets the multibuffering attributes that apply to all buffers associated with the given window. This is currently limited to the update\_hint.

*XmbufGetBufferAttributes* gets the attributes for the indicated buffer. Returns non-zero on success and zero if an error occurs.

*XmbufChangeBufferAttributes* sets the attributes for the indicated buffer. This is currently limited to the event\_mask.

*XmbufGetScreenInfo* gets the parameters controlling how mono and stereo windows may be created on the screen of the given drawable. The numbers of sets of visual and depths are returned in nmono\_return and nstereo\_return. If nmono\_return is greater than zero, then mono\_info\_return is set to the address of an array of *XmbufBufferInfo* structures describing the various visuals and depths that may be used. Otherwise, mono\_info\_return is set to NULL. Similarly, stereo\_info\_return is set according to nstereo\_return. The storage returned in mono\_info\_return and stereo\_info\_return may be released by *XFree*. If no errors are encountered, non-zero will be returned.

*XmbufCreateStereoWindow* creates a stereo window in the same way that *XCreateWindow* creates a mono window. The buffer ids for the left and right buffers are returned in left\_return and right\_return, respectively. If an extension error handler that returns is installed, *None* will be returned if the extension is not available on this display.

## PREDEFINED VALUES

Update\_action field:

*MultibufferUpdateActionUndefined*  
*MultibufferUpdateActionBackground*  
*MultibufferUpdateActionUntouched*

*MultibufferUpdateActionCopied*

Update\_hint field:

*MultibufferUpdateHintFrequent*

*MultibufferUpdateHintIntermittent*

*MultibufferUpdateHintStatic*

Valuemask fields:

*MultibufferWindowUpdateHint*

*MultibufferBufferEventMask*

Mono vs. stereo and left vs. right:

*MultibufferModeMono*

*MultibufferModeStereo*

*MultibufferSideMono*

*MultibufferSideLeft*

*MultibufferSideRight*

Clobber state:

*MultibufferUnclobbered*

*MultibufferPartiallyClobbered*

*MultibufferFullyClobbered*

Event stuff:

*MultibufferClobberNotifyMask*

*MultibufferUpdateNotifyMask*

*MultibufferClobberNotify*

*MultibufferUpdateNotify*

*MultibufferNumberEvents*

*MultibufferBadBuffer*

*MultibufferNumberErrors*

## **BUGS**

This manual page needs more work.

## **SEE ALSO**

*Extending X for Double Buffering, Multi-Buffering, and Stereo*