XMClib
C Language XMC Interface

Version 0.12
February 12, 1998

John Bazik
Computer Science Department
Brown University
jsb@cs.brown.edu

1. Introduction

A network-transparent window system, such as the X Window System, separates application programs from graphical display hardware and input devices by a protocol stream carried over a local or remote data connection. All program input and output is transmitted via this stream. A protocol multiplexor is an intermediary process that allows each application program's protocol stream to be directed to multiple display servers. The application program need not be aware that its input is received from and its output is sent to more than one display. In fact, neither the applications nor the display servers need to be modified in any way.

The shared environment that is created by a multiplexor-enhanced window session requires some dynamic controls to realize its full potential. Following the model of the X Window System, these controls are defined as a communication protocol: X Multiplexor Control Protocol (XMCP). Multiplexor-clients send requests to the multiplexor to add or remove a display from the session, change the way that user inputs are interpreted, and create or change telepointers. The multiplexor uses the protocol to provide information requested by multiplexor-clients and to inform them of changes to the session.

Like X, XMCP is designed to provide as much mechanism as a multiplexor can provide, while imposing as little policy as possible. The way in which application programs and their resources are shared in a multiplexed window session is entirely under the control of its multiplexor clients, just as the look and interaction of an X session is dictated by its window manager and other X clients.

For the programmer, XMClib provides a procedural library interface to the protocol. Like Xlib, it is a low-level interface that closely reflects the features of the protocol, and imposes no policies that are not protocol invariants.

1.1. Overview

Overview goes here.

1.2. Errors

1.3. Standard Header Files

- `<xmclib.h>`

2. Display Functions

Before a multiplexor client can control a session, it must establish a connection to the multiplexor. Connections come in various flavors, depending upon the credentials used to establish them. A multiplexor maintains a list of cookies which may be used to open a multiplexor control connection. Some of these cookies are known by the multiplexor at startup time, others may be added by multiplexor clients. If authentication is not explicitely turned on, unauthenticated connections from the local host are permitted. Access lists, as defined in the X Protocol, have no meaning for multiplexor control connections.

Cookies are name, data pairs that identify the possessor as an authenticated client. The mechanism is the same as that used by X servers, and multiplexor control clients may use the same authority database as X clients and servers.

```
typedef struct {
    unsigned short namelen;
    char * name;
    unsigned short datalen;
    char *data;
}XmcCookie;
```

2.1. Opening a Connection

To open a connection to the multiplexor that controls the session, use `XmcOpen`.

Mux *XmcOpen(*display, cookie*)
    char **display*;
    XmcCookie **cookie;*
display Specifies the multiplexor to which to connect.
cookie Xauth cookie or null.

The encoding of the display name is the same as that used in XLib, the C Language X Interface [fn]. It is interpreted in a way that never conflicts with the interpretation used by X servers. That is, a multiplexor that presents both X and XMC network services can safely be identified to both protocols by the same display name.

If the multiplexor is running with authentication turned on, it will require a valid cookie to complete the connection. If no cookie is provided to this function, then if the environment variable XMAUTHORITY is set, the file it names is searched for a valid cookie. If XMAUTHORITY is not set and a file named `.XMauthority` exists in the user's home directory, it is searched, otherwise the standard X authority mechanism is used. If no cookie is found, a connection is attempted without a cookie.

Different cookies afford their presenters different levels of access to multiplexor control functions. These are described in the section on authentication.

TptrId DefaultTptr(*muxp*)
    Mux **muxp*;
muxp Specifies the multiplexor connection.

Returns the id of the default telepointer. There is always a default telepointer, even if the multiplexor was started in "no telepointer" mode. The default telepointer cannot be destroyed.

int ConfigMode(*muxp*)
    Mux **muxp*;
muxp Specifies the multiplexor connection.

Returns the current config mode of the multiplexor. This value can only be `ConfigAllow` or `ConfigDelay`. Initially it is always set to `ConfigAllow` and can only be changed by a call to XmcDelayConfig or XmcAllowConfig.

XmcClose(*muxp*)
    Mux **muxp*;
muxp Specifies the multiplexor connection.

Close the connection to the multiplexor represented by *muxp*.

2.2. Registering the Session

Multiplexors may be able to identify themselves to one or more directory services, which aid users in locating shared sessions in progress.

```
typedef struct {
    int family;
    int length;
    char *address;
}XmcHostAddress;
```

RegId XmcRegister(*muxp*, *host*, *port*, *name*, *url*, *about*)
    Mux **muxp*;
    XmcHostAddress **host*;
    int *port*;
    char **name*;
    char **url*;
    char **about*;

muxp	Specifies the multiplexor connection.
host	Host of session directory.
port	Port of session directory.
name	Name of session.
url	URL where authority information is located.
about	Description of session.

Register this session with the session directory located at (*host*, *port*), giving it *name*, directing prospective participants to connection information located at *url* and providing a brief description, *about*.

XmcUnregister(*muxp*, *id*)
    Mux **muxp*;
    RegId *id*;

muxp	Specifies the multiplexor connection.
id	Registration.

Remove a session registration.

2.3. Authorization and Permissions

The functions in this section use a bitmask type, PMask, which defines what actions an XMC client may take based on what cookie it used to establish its connection.

```
/* permission mask bits */
#define BaseAuth        0x0000  /* */
#define AddMine         0x0002  /* may perform a DisplayAdd to own display */
#define AddAny          0x0003  /* may perform a DisplayAdd to any display */
#define FloorMine       0x0008  /* may change the input mode of own display to floor */
#define FloorAny        0x000c  /* may change the input mode of any display to floor */
#define SeatMine        0x0020  /* may change the input mode of own display to seat */
#define SeatAny         0x0030  /* may change the input mode of any display to seat */
#define GrabMine        0x0080  /* may grab keyboard or pointer on own display */
#define GrabAny         0x00c0  /* may grab keyboard or pointer on any display */
#define TptrMine        0x0200  /* may assign telepointer to own display */
#define TptrAny         0x0300  /* may assign telepointers to any display */
#define SelectMine      0x0800  /* may share selections on own display */
#define SelectAny       0x0c00  /* may share selections on any display */
#define LocalAuth       0x0aaa  /* may do anything, but only on own display */
#define SuperAuth       0x1000  /* may do anything */
```

XmcSetAuth(*muxp*, *cookie*, *perms*)
    Mux **muxp*;
    Cookie **cookie*;
    PMask *perms*;

muxp	Specifies the multiplexor connection.
cookie	Cookie.
perms	Bitmask of permissions.

Establish a cookie with the given permissions, or replace the permissions of an existing cookie.

int XmcGetAuth(*muxp*, *cookie*, *permsp*)
    Mux **muxp*;
    Cookie **cookie*;
    PMask **permsp*

muxp	Specifies the multiplexor connection.
cookie	Cookie.
permsp	Bitmask of permissions.

Get permissions bitmask associated with cookie.

2.4. Managing Multiple Displays

The essential feature of a multiplexor is its ability to show a single virtual X session on multiple displays. The functions in this section are concerned with managing those displays.

```
typedef struct {
    int mode;
    int screen;
    int display;
    int family;
    int addrlen;
    char *addrp;
    char *tag;
    TptrId tpID;
    XId window;          /* real server window id of top-level virtual root window */
}XmcDispInfo;
```

XmcAddDisplay(*muxp, display, cookie, geom_str, window, mode, tag, tptrID*)

```
Mux *muxp;
char *display;
XmcCookie *cookie;
char *geom_str;
XId window;
int mode;
char *tag;
TptrId tptrID;
```

muxp	Specifies the multiplexor connection.
display	Display.
cookie	Xauth cookie or null.
geom_str	Geometry string or null.
window	Real server window id of top-level window, or null.
mode	Input mode.
tag	Tag.
tptrID	Telepointer or null.

Add a display.

XmcSetDisplayTag(*muxp, dispID, tag*)

```
Mux *muxp;
DispId dispID;
char *tag;
```

muxp	Specifies the multiplexor connection.
dispID	Display.
tag	Tag.

Set the tag string associated with the given display.

XmcDispInfo *XmcQueryDisplay(*muxp, dispID*)

```
Mux *muxp;
DispId dispID;
```

muxp	Specifies the multiplexor connection.
dispID	Display.

Query a display.

DispId *XmcListDisplays(*muxp, ndisp*)

```
Mux *muxp;
int *ndisp;
```

muxp	Specifies the multiplexor connection.
ndisp	Returns the number of displays listed.

List displays.

XmcFreeDisplayList(*dispIDs*)

```
DispId *dispIDs;
```

dispIDs	List to free.

Free display list.

DispId *XmcListDisplaysWithInfo(*muxp*, *ndisp_return*, *info_return*)
    Mux **muxp*;
    int **ndisp_return*;
    XmcDispInfo **info_return*;
muxp Specifies the multiplexor connection.
ndisp Returns the number of displays listed.
infop List of display information.

Not implemented.

XmcFreeDisplayInfo(*dispinfo*, *ndisp*)
    XmcDispInfo **dispinfo*;
    int *ndisp*;
dispinfo List to free.
ndisp Number of elements in the list.

Free display info list.

XmcDropDisplay(*muxp*, *dispID*)
    Mux **muxp*;
    DispId *dispID*;
muxp Specifies the multiplexor connection.
dispID Display.

Drop a display.

2.5. Synchronizing the Displays

XmcSync(*muxp*)
    Mux **muxp*;
muxp Specifies the multiplexor connection.

Synchronize all displays.

2.6. Virtual Configuration

The virtual configuration is the set of X resources and settings seen by X client applications, but not created or set by any of them. The virtual configuration includes all of the information sent to an X client by xmx in an X server connection block, as well as the list of supported protocol extensions. It is sometimes helpful to be able to control the configuration seen by clients. One example is a session begun by several highly configured machines, to which a less well configured one is added later. By limiting the initial configuration to only those capabilities needed in the session, the display added later will encounter no problem joining.

The virtual configuration is always available for query, but may only be set before it is *fixed*, which occurs after the initial set of X servers have been contacted, but before virtual root windows have been established on them. No X client applications may be connected to the multiplexor before the virtual configuration is fixed. Changes to the virtual configuration must be compatible with any X servers to which the multiplexor is connected, otherwise they are rejected. If no servers are connected, all settings are accepted, but the resulting virtual configuration may not be compatible with any X servers that are later added.

The X Protocol makes provision for the support of multiple color models. These models are defined by a depth, a class of visual and several other attributes. For most purposes, the depth and class are all that matter, so they are considered together here as a depth-type. Two examples of this are 24-bit TrueColor and 8-bit PseudoColor. The XMC protocol supports the specification of which depth-types the virtual root window must support. If specified, no other depth-types are made part of the virtual configuration.

[mention defaults]

```
typedef struct {
    int depth;
    unsigned int visualclass;
}XmcDepthType;
```

These routines are not yet implemented.

XmcSetConfig(*muxp, width, height, ndepthtypes, depthtypesp, nextensions, extensions*)

```
    Mux *muxp;
    unsigned int width;
    unsigned int height;
    int ndepthtypes;
    XmcDepthType *depthtypesp;
    int nextensions;
    char **extensions;
```

muxp	Specifies the multiplexor connection.
width	Width in pixels of virtual screen. If either width or height is zero, neither is specified.
height	Height in pixels of virtual screen.
ndepthtypes	Number of depth, visualclass pairs supported by this screen. If less than one, not specified.
depthtypesp	Array of depth, visualclass pairs. The first is the default for this screen.
nextensions	Number of extensions. If negative, not specified.
extensions	List of extension names.

Not implemented.

int XmcGetConfig(*muxp, widthp, heightp, ndepthtypesp, depthtypesp, nextensionsp, extensionsp*)

```
    Mux *muxp;
    unsigned int *widthp;
    unsigned int *heightp;
    int *ndepthtypesp;
    XmcDepthType **depthtypesp;
    int *nextensionsp;
    char ***extensions;
```

muxp	Specifies the multiplexor connection.
widthp	Width in pixels of virtual screen.
heightp	Height in pixels of virtual screen.
ndepthtypesp	Number of depth, visualclass pairs supported by this screen.
depthtypesp	Array of depth, visualclass pairs.
nextensionsp	Number of extensions supported.
extensionsp	List of extension names.

Not implemented.

XmcFreeDepthTypeInfo(*depthtypesp, ndepthtypes*)

```
    XmcDepthType *depthtypesp;
    int ndepthtypes
```

depthtypesp	XmcDepthType list to free.
ndepthtypes	Number of elements in list.

Free storage allocated for an XmcDepthType list.

XmcFreeExtensionList(*extensionsp*, *nextensions*)
    char ***extensionsp*;
    int *nextensions*
extensionsp Extension list to free.
nextensions Number of elements in list.

Free storage allocated for an extension name list.

XmcSetConfigMode(*muxp*, *mode*)
    Mux **muxp*;
    int *mode*
muxp Specifies the multiplexor connection
mode One of `ConfigAllow`, `ConfigDelay` or `ConfigAny`.

If the virtual configuration is not yet fixed, the config mode dictates when it will be. Mode `ConfigAllow` causes the configuration to be fixed at the next opportunity, which is when one or more displays are added. Mode `ConfigDelay` causes the multiplexor to postpone establishing the virtual configuration until the config mode is again set to `ConfigAllow`. This routine has the side effect of establishing a preference for the calling application, so that subsequent calls to XmcAddDisplay will fail if the current config mode does not match that preference. Mode `ConfigAny` indicates that the calling application has no preference. The default multiplexor mode is `ConfigAllow` and by default applications have no preference.

3. Input Control

3.1. Input Mode

XmcChangeInputMode(*muxp*, *dispID*, *mode*)
    Mux **muxp*;
    DispId *dispID*;
    int *mode*;
muxp Specifies the multiplexor connection.
dispID Display.
mode Mode.

Change input mode.

3.2. Events

A multiplexor client sends requests to the multiplexor through the connection established in the XmcOpen function. The multiplexor uses that same connection to notify the multiplexor client of significant events. Some events are caused by requests, others by user input. This section describes those events.

3.2.1 Event Structures

```
typedef union {
    int type;
    XmcDisplayIdEvent display;
    XmcTptrAssignEvent tptrassign;
    XmcTptrIdEvent tptrvis;
    XmcButtonEvent button;
    XmcKeyEvent key;
    XmError error;
}XmcEvent;
```

```
typedef struct {
    int type;               /* Event type */
    unsigned long serial;   /* # of last request processed by multiplexor */
    DispId id;              /* Display the event was  read from */
}XmcDisplayIdEvent;
```

3.2.2 Event Masks

NoEventMask	No events wanted
DisplayMask	Display add, drop or tag events wanted
InputModeMask	Input mode change events wanted
GrabPointerMask	Grab or ungrab pointer events wanted
GrabKeyboardMask	Grab or ungrab keyboard events wanted
SelectionMask	Share or unshare selections events wanted
TptrAssignMask	Telepointer assignment events wanted
TptrVisibilityMask	Telepointer show or hide events wanted
SeatKeyPressMask	Keyboard down events wanted
SeatKeyReleaseMask	Keyboard up events wanted
SeatButtonPressMask	Pointer button down events wanted
SeatButtonReleaseMask	Pointer button up events wanted

3.2.3 Display Events

```
typedef XmcDisplayIdEvent XmcDisplayInEvent;
typedef XmcDisplayIdEvent XmcDisplayRefusedEvent;
typedef XmcDisplayIdEvent XmcDisplayOutEvent;
typedef XmcDisplayIdEvent XmcDisplayTagEvent;

typedef XmcDisplayIdEvent XmcModeFloorEvent;
typedef XmcDisplayIdEvent XmcModeSeatEvent;
typedef XmcDisplayIdEvent XmcModeViewEvent;

typedef XmcDisplayIdEvent XmcPointerGrabEvent;
typedef XmcDisplayIdEvent XmcPointerNoGrabEvent;
typedef XmcDisplayIdEvent XmcPointerUngrabEvent;
typedef XmcDisplayIdEvent XmcKeyboardGrabEvent;
typedef XmcDisplayIdEvent XmcKeyboardNoGrabEvent;
typedef XmcDisplayIdEvent XmcKeyboardUngrabEvent;

typedef XmcDisplayIdEvent XmcShareSelectionsEvent;
typedef XmcDisplayIdEvent XmcUnshareSelectionsEvent;

typedef struct {
    int type;
    unsigned long serial;
    DispId id;
    TptrId tptrID;
}XmcTptrEvent;
typedef XmcTptrEvent XmcTptrAssignEvent;
typedef XmcTptrEvent XmcTptrHideEvent;
typedef XmcTptrEvent XmcTptrShowEvent;
```

```
typedef struct {
    int type;
    unsigned long serial;
    unsigned int mode;
}XmcConfigModeEvent;
```

3.2.4 X Events

```
typedef struct {
    int type;
    unsigned long serial;
    DispId id;
    int send_event;
    XId window;
    XId subwindow;
    unsigned long int time;
    short int x, y;
    short int x_root, y_root;
    unsigned int state;
    unsigned int button;
}XmcButtonEvent;
typedef XmcButtonEvent XmcButtonPressedEvent;
typedef XmcButtonEvent XmcButtonReleased Event;

typedef struct {
    int type;
    unsigned long serial;
    DispId id;
    int send_event;
    XId window;
    XId subwindow;
    unsigned long int time;
    short int x, y;
    short int x_root, y_root;
    unsigned int state;
    unsigned int keycode;
}XmcKeyEvent;
typedef XmcKeyEvent XmcKeyPressedEvent;
typedef XmcKeyEvent XmcKeyReleased Event;
```

3.3. Selecting Events

XmcSetEventMask(*muxp*, *mask*)
    Mux **muxp*;
    Emask *mask*;

muxp	Specifies the multiplexor connection.
mask	Event mask.

Set event mask.

int XmcGetEventMask(*muxp, mask*)
    Mux **muxp*,
    Emask **mask*,
muxp Specifies the multiplexor connection.
mask Returns the event mask.

Get the event mask.

XmcSetXEventMask(*muxp, window, mask*)
    Mux **muxp*;
    XId *window*;
    XEmask *mask*;
muxp Specifies the multiplexor connection.
window Window.

Set X mask.

int XmcGetXEventMask(*muxp, window, maskp*)
    Mux **muxp*;
    XId *window*;
    XEmask **maskp*;
muxp Specifies the multiplexor connection.
window Window.
maskp Returns mask.

Get X mask.

int XmcEventsQueued(*muxp, mode*)
    Mux **muxp*;
    int *mode*;
muxp Specifies the mulitplexor connection.
mode {XmcQueuedAlready, XmcQueuedAfterFlush, XmcQueuedAfterReading}

How many events are queued?

int XmcGetEvent(*muxp, event*)
    Mux **muxp*;
    XmcEvent **event*;
muxp Specifies the multiplexor connection.
event Returns the next event.

Get next event.

3.4. Grabbing Input

The multiplexor's virtual root window normally behaves like a good X client window, ceding control over local window management to the local window manager and user. However, because a whole X session exists within that window, it is sometimes useful to steal control over the pointer or keyboard away from the local session. By grabbing control in this way, the shared window manager receives all events that the user initiated, unfiltered by the local window manager. The use of this feature can cause surprising effects to local X sessions and should be used carefully.

The pointer or keyboard grab is only in effect when the user's focus is in the virtual root window. Actual X protocol grabs are not guaranteed to happen. When a grab is in effect, each time the focus is acquired by the virtual root window, the multiplexor will attempt a grab using X protocol. If the grab is unsuccessful, no further attempt is made until the focus is lost and then acquired again. Clients wishing to provide feedback to users on the state of the grab

should select for Grab events.

XmcGrabPointer(*muxp*, *dispId*)
    Mux **muxp*;
    DispId *dispId*;
muxp Specifies the multiplexor connection.
dispId Display.

Grab input.

XmcUngrabPointer(*muxp*, *dispId*)
    Mux **muxp*;
    DispId *dispId*;
muxp Specifies the multiplexor connection.
dispId Display.

Release an input grab.

XmcGrabKeyboard(*muxp*, *dispId*)
    Mux **muxp*;
    DispId *dispId*;
muxp Specifies the multiplexor connection.
dispId Display.

Grab input.

XmcUngrabKeyboard(*muxp*, *dispId*)
    Mux **muxp*;
    DispId *dispId*;
muxp Specifies the multiplexor connection.
dispId Display.

Release an input grab.

3.5. Sharing Selections

Though a shared virtual X session is a world unto itself, isolated from the real X sessions through which it is brought to its participants, some interaction with those real sessions can be helpful. One particularly useful feature is the ability to cut and paste between an individual user's X session and the shared session. The X protocol offers selections as a mechanism for this kind of inter-client communication. The multiplexor implements selections within its virtual session, but it can make them blend seamlessly with the selections of none, any or all real X sessions. Note that among all real and virtual X sessions that are sharing selections through this mechanism, there is never more than one of a particular selection. This should be used carefully as it may surprise users to lose control of their selections.

XmcShareSelections(*muxp*, *dispId*)
    Mux **muxp*;
    DispId *dispId*;
muxp Specifies the multiplexor connection.
dispId Display.

Share selections between the virtual X session and the real X session on the given display.

XmcUnshareSelections(*muxp, dispId*)
    Mux **muxp*;
    DispId *dispId*;
muxp Specifies the multiplexor connection.
dispId Display.

Do not share selections between the virtual X session and the real X session on the given display.

3.6. Handling the Output Buffer

XmcFlush(*muxp*)
    Mux **muxp*;
muxp Specifies the multiplexor connection.

Flush output.

3.7. Handling Errors

XmcSetErrorHandler(*handler*))()
    int (**handler*)(Mux, XmcError *);
handler Specifies the error handler.

XmcSetIOErrorHandler(*handler*))()
    int (**handler*)(Mux);
handler Specifies the IO error handler.

4. Telepointers

4.1. Creating and Freeing Telepointers

TptrId XmcCreatePixmapTptr(*muxp, displaymask, source, mask, x, y, fgcolor, bgcolor*)
    Mux **muxp*;
    unsigned int *displaymask*;
    XId *source*;
    XId *mask*;
    int *x*;
    int *y*;
    XmcColor **fgcolor*;
    XmcColor **bgcolor*;
muxp Specifies the multiplexor connection.
displaymask Specifies the way the telepointer will be displayed.
source A pixmap in the shared session that specifies the telepointer's shape.
mask A pixmap in the shared session that specifies the bits from *source* to be displayed.
x
y Specifies the telepointer's hot-spot relative to the origin of the pixmap.
fgcolor The RGB values for the foreground of the source.
bgcolor The RGB values for the background of the source.

Not implemented.

TptrId XmcCreateGlyphTptr(*muxp, displaymask, src_font, mask_font, src_char, mask_char, fgcolor, bgcolor*)
    Mux **muxp*;
    unsigned int *displaymask*;
    XId *src_font*;
    XId *mask_font*;
    unsigned int *src_char*;
    unsigned int *mask_char*;
    XmcColor **fgcolor*;
    XmcColor **bgcolor*;

muxp	Specifies the multiplexor connection.
displaymask	Specifies the way the telepointer will be displayed.
src_font	A font in the shared session that contains the source glyph.
mask_font	A font in the shared session that contains the mask glyph.
src_char	The character glyph for the source.
mask_char	The character glyph for the mask.
fgcolor	The RGB values for the foreground of the source.
bgcolor	The RGB values for the background of the source.

Not implemented.

XmcDestroyTptr(*muxp, tptrID*)
    Mux **muxp;*
    TptrId *tptrID*;

muxp	Specifies the multiplexor connection.
tptrID	Specifies the telepointer to destroy.

Not implemented.

4.2. Managing Telepointers

Each X server represents a potential source of input events to the multiplexor. Pointer movement events are only generated by displays that hold the floor. Any telepointer may be assigned to any display, and that display's pointer movements will then be reflected in the movement of that telepointer. A telepointer may be assigned to any number of displays, but each display is assigned only one telepointer. If no display assigned to a telepointer has the floor, the telepoiner is not displayed.

XmcAssignTptr(*muxp, dispID, tptrID*)
    Mux **muxp*;
    DispId *dispID*;
    TptrId *tptrID*;

muxp	Specifies the multiplexor connection.
dispID	Display.
tptrID	Telepointer.

Assign a telepointer.

XmcHideTptr(*muxp, tptrID*)
    Mux **muxp*;
    TptrId *tptrID*;

muxp	Specifies the multiplexor connection.
tptrID	Telepointer.

Hide a telepointer.

XmcShowTptr(*muxp*, *tptrID*)
    Mux **muxp*;
    TptrId *tptrID*;

muxp	Specifies the multiplexor connection.
tptrID	Telepointer.

Show telepointer.

5. Connection Close