Translations

VirtualBindings

VirtualBindings (library call)

NAME

VirtualBindings – Bindings for virtual mouse and key events

DESCRIPTION

The Motif reference pages describe key translations in terms of *virtual bindings*, based on those described in the *Motif Style Guide*.

Bindings for osf Keysyms

Keysym strings that begin with `osf` are not part of the X server's keyboard mapping. Instead, these keysyms are produced on the client side at run time. They are interpreted by the routine **XmTranslateKey**, and are used by the translation manager when the server delivers an actual key event. For each application, a mapping is maintained between `osf` keysyms and keysyms that correspond to actual keys. This mapping is based on information obtained at application startup from one of the following sources, listed in order of precedence:

- The **XmNdefaultVirtualBindings** resource from Display.
- A property on the root window, which can be set by **mwm** on startup, or by the **xmbind** client, or on prior startup of a Motif application.
- The file `.motifbind` in the user's home directory.
- A set of bindings based on the vendor string and optionally the vendor release of the X server. Motif searches for these bindings in the following steps:
  1. If the file `xmbind.alias` exists in the user's home directory, Motif searches this file for a pathname associated with the vendor string or with the vendor string and vendor release. If it finds such a pathname and if that file exists, Motif loads the bindings contained in that file.
  2. If it has found no bindings, Motif next looks for the file `xmbind.alias` in the directory specified by the environment variable **XMBINDDIR**, if **XMBINDDIR** is set, or in the directory `/usr/lib/Xm/bindings` if **XMBINDDIR** is not set. If this file exists Motif searches it for a pathname associated with the vendor string or with the vendor string and vendor release. If it finds such a pathname and if that file exists, Motif loads the bindings contained in that file.
  3. If it still has found no bindings, Motif loads a set of hard–coded fallback bindings.

The `xmbind.alias` file contains zero or more lines of the following form:

`"`vendor_string `[` vendor_release `]"` bindings_file

where *vendor_string* is the X server vendor name as returned by the X client **xdpyinfo** or the Xlib function **XServerVendor**, and must appear in double quotes. If *vendor_release* is included, it is the X server vendor release number as returned by the X client **xdpyinfo** or the Xlib function **XVendorRelease**, and must also be contained within the double quotes separated by one space from *vendor_string*. The *vendor_release* argument is provided to allow support of changes in keyboard hardware from a vendor, assuming that the vendor increments the release number to flag such changes. Alternatively, the vendor may simply use a unique vendor string for each different keyboard.

The *bindings_file* argument is the pathname of the file containing the bindings themselves. It can be a relative or absolute pathname. If it it is a relative pathname, it is relative to the location of the `xmbind.alias` file.

Comment lines in the `xmbind.alias` file begin with ! (exclamation point).

The bindings found in either the `.motifbind` file or the vendor mapping are placed in a property on the root window. This property is used to determine the bindings for subsequent Motif applications.

On startup **mwm** attempts to load the file `.motifbind` in the user's home directory. If this is unsuccessful, it loads the vendor bindings as described previously. It places the bindings it loads in a property on the root window for use by subsequent Motif applications.

The **xmbind** function loads bindings from a file if that file is specified on the command line. If no file is specified on the command line, it attempts to load the file `.motifbind` in the user's home directory. If this fails, it loads the vendor bindings as described previously. It places the bindings it loads in a property on the root window for use by subsequent Motif applications.

The format of the specification for mapping `osf` keysyms to actual keysyms is similar to that of a specification for an event translation. (See below) The syntax is specified (and below) here in EBNF notation using the following conventions:

`[`a`]` `Means either nothing or` a

`{`a`}` `Means zero or more occurrences of` a

(a|b) Means either a or b.

Terminals are enclosed in double quotation marks.

The syntax of an osf keysym binding specification is as follows:

```
binding_spec    =       {line "\n"} [line]
line            =       virtual_keysym ":" list_of_key_event
list_of_key_event=      key_event { "," key_event}
key_event       =       {modifier_name} "<Key>" actual_keysym
virtual_keysym  =       keysym
actual_keysym   =       keysym
keysym          =       A valid X11 keysym name that is
                        mapped by XStringToKeysym
```

As with event translations, more specific event descriptions must precede less specific descriptions. For example, an event description for a key with a modifier must precede a description for the same key without the same modifier.

Following is an example of a specification for the *defaultVirtualBindings* resource in a resource file:

```
*defaultVirtualBindings: \
        osfBackSpace:       <Key>BackSpace       \n\
        osfInsert:       <Key>InsertChar      \n\
        osfDelete:       <Key>DeleteChar      \n\
...
        osfLeft:       <Key>left, Ctrl<Key>H
```

The format of a .motifbind file or of a file containing vendor bindings is the same, except that the binding specification for each keysym is placed on a separate line. The previous example specification appears as follows in a .motifbind or vendor bindings file:

```
osfBackSpace:       <Key>BackSpace
osfInsert:       <Key>InsertChar
osfDelete:       <Key>DeleteChar
...
osfLeft:       <Key>left, Ctrl<Key>H
```

The following table lists the fixed fallback default bindings for osf keysyms.

Fallback Default Bindings for osf Keysyms	
osf Keysym	*Fallback Default Binding*
osfActivate:	KeyKP_Enter Key*Execute*
osfAddMode:	*Shift*<*Key*>*F8*
osfBackSpace:	Key*BackSpace*
osfBeginLine:	KeyHome Key*Begin*
osfCancel:	KeyEscape Key*Cancel*
osfClear:	Key*Clear*
osfCopy:	*unbound*
osfCut:	*unbound*
osfDelete:	Key*Delete*
osfDeselectAll:	*unbound*
osfDown:	Key*Down*
osfEndLine:	Key*End*
osfHelp:	KeyF1 Key*Help*
osfInsert:	Key*Insert*
osfLeft:	Key*Left*
osfLeftLine:	*unbound*
osfMenu:	*Shift*KeyF10 Key*Menu*
osfMenuBar:	KeyF10 ShiftKey*Menu*
osfNextMinor:	*unbound*
osfPageDown:	Key*Next*
osfPageLeft:	*unbound*
osfPageRight:	*unbound*
osfPageUp:	Key*Prior*
osfPaste:	*unbound*
osfPrimaryPaste:	*unbound*
osfPriorMinor:	*unbound*
osfReselect:	*unbound*
osfRestore:	*unbound*
osfRight:	Key*Right*
osfRightLine:	*unbound*
osfSelect:	Key*Select*
osfSelectAll:	*unbound*
osfSwitchDirection:	*Alt*KeyReturn AltKey*KP_Enter*
osfUndo:	Key*Undo*
osfUp:	Key*Up*

Changes in the Handling of Shifted Keys

In conjunction with MIT X11R5 Patch 24, this version of Motif introduces a change in the way that keys involving the <Shift> modifier are processed. This change allows the numeric keypad to be used to generate numbers using the standard X mechanisms. Since the default behavior is now to honor the xmodmap keymap bindings, translations and virtual key bindings that use <Shift> may behave differently. A common symptom is that unshifted keypad and function keys (with or without other modifiers) produce the expected results, but shifted ones do not.

To obtain the old behavior you can remove the shifted interpretation from problematic keys using the **xmodmap** utility. Each entry in a **xmodmap** keymap table contains up to four keysym bindings. The second and fourth keysyms are for shifted keys. If an expression contains only two keysyms, simply remove the second keysym. If an entry contains three or more keysyms, replace the second keysym with **NoSymbol** and remove the fourth keysym.

Action Translations

The translation table syntax used by Motif is completely specified in the X11R5 Toolkit Intrinsics Documentation. For the complete syntax description, and for general instructions about writing or modifying a translation table, please refer to this document. A brief summary of the translation table format, however, is included below.

The syntax is defined as in the binding syntax specification above. Informal descriptions are contained in angle brackets (<>).

```
TranslationTable=       [ directive ] { production }
directive       =       ( "#replace" | "#override" | "#augment") "\n"
production      =       lhs ":" rhs "\n"
lhs             =       ( event | keyseq) {"," ( event | keyseq) }
keyseq          =       """ keychar { keychar } """
keychar         =       ( "[caret ]" | "$" | "\\") <ISO Latin 1 character>
event           =       [ modifier_list ] "<" event_type ">" [ count ] {detail}
```

```
modifier_list   =       ( ["!"][":"] { modifier } | "None")
modifier        =       [ "~" ] ( "@" <keysym> | <name from table below>)
count           =       "(" <positive integer> [ "+" ] ")"
rhs             =       { action_name "(" [params] ")" }
params          =       string { "," string }
```

The *string* field need not be quoted unless it includes a space or tab character, or any comma, newline, or parenthesis. The entire list of string values making up the *params* field will ba passed to the named action routine.

The *details* field may be used to specify a keysym that will identify a particular key event. For example, `Key` is the name of a type of event, but it must be modified by the *details* field to name a specific event, such as `Key`*A*.

Modifier Names The modifier list, which may be empty, consists of a list of modifier keys that must be pressed with the key sequence. The modifier keys may abbreviated with single letters, as in the following list of the familiar modifiers:

s

Shift

c or [caret]

Ctrl (Control)

m or $

Meta

a

Alt

Other modifiers are available, such as "Mod5" and "Button2." These have no abbreviation (although the "Button" modifiers may be abbreviated in combination with events, as outlined below). If a modifier list has no entries, and is not "None", it means the position of the modifier keys is irrelevant. If modifiers are listed, the designated keys must be in the specified position, but the unlisted modifier keys are irrelevant. If the list begins with an exclamation point (!), however, the unlisted modifiers may not be asserted. In addition, if a modifier name is preceded by a tilde (~), the corresponding key must *not* be pressed.

If a modifier list begins with a colon (:), X tries to use the standard modifiers (Shift and Lock), if present, to map the key event code into a recognized keysym.

Event Types These are a few of the recognized event types.

Key or KeyDown

A keyboard key was pressed.

KeyUp

A keyboard key was released.

BtnDown

A mouse button was pressed.

BtnUp

A mouse button was released.

Motion

The mouse pointer moved.

Enter

The pointer entered the widget’s window.

Leave

The pointer left the widget's window.

FocusIn

The widget has received focus.

FocusOut

The widget has lost focus.

There are some event abbreviations available. For example, `Btn1Motion` is actually a "Motion" event, modified with the "Button1" modifier (*Button1*<*Motion*>). Similarly, `Btn3Up` is actually a "BtnUp" event with the "Button3" modifier. These abbreviations are used extensively in the Motif translation tables.

RELATED

xmbind(1)