Section 4 - Mrm Clients

This page describes the format and contents of each reference page in Section 4, which covers the Motif clients.

Name

Client – a brief description of the client.

Syntax

This section describes the command-line syntax for invoking the client. Anything in **bold** should be typed exactly as shown. Items in *italics* are parameters that should be replaced by actual values when you enter the command. Anything enclosed in brackets is optional.

Availability

This section appears for functions that were added in Motif 2.0 or later.

Description

This section explains the operation of the client. In some cases, additional descriptive sections appear later on in the reference page.

Options

This section lists available command-line options.

Environment

If present, this section lists shell environment variables used by the client. This section does not list the DISPLAY and XENVIRONMENT variables, which are used by all clients. These variables are used as follows:

DISPLAY
: To get the default display name (specifically, the host, server/display, and screen). The DISPLAY variable typically has the form:

    hostname:*server.screen*

XENVIRONMENT
: To get the name of a resource file containing host-specific resources. If this variable is not set, the resource manager will look for a file called .Xdefaults-**hostname** (where **hostname** is the name of a particular host) in the user's home directory.

Bugs

If present, this section lists any problems that could arise when using the client.

See Also

This section refers you to related clients, functions, or widget classes. The numbers in parentheses following each reference refer to the sections of the book in which they are found.

Name

mwm – the Motif Window Manager (*mwm*).

Syntax

mwm [*options*]

Description

The Motif Window Manager, *mwm*, provides all of the standard window management functions. It allows you to move, resize, iconify/deiconify, maximize, and close windows and icons, focus input to a window or icon, and refresh the display. *mwm* provides much of its functionality via a frame that (by default) is placed around every window on the display. The *mwm* frame has the three-dimensional appearance characteristic of the OSF/Motif graphical user interface.

The rest of this reference page describes how to customize *mwm*. It does not provide information on using *mwm*. For information on using the window manager, see Volume 3, *X Window System User's Guide, Motif Edition*.

Options

-display [*host]*:*server*[.*screen*]

Specifies the name of the display on which to run *mwm*. *host* is the hostname of the physical display, *server* specifies the server number, and *screen* specifies the screen number. Either or both of the *host* and *screen* elements can be omitted. If *host* is omitted, the local display is assumed. If *screen* is omitted, screen 0 is assumed (and the period is unnecessary). The colon and (display) *server* are necessary in all cases.

-multiscreen

Specifies that *mwm* should manage all screens on the display. The default is to manage only screen 0. You can specify an alternate screen by setting the DISPLAY environment variable or using the -display option. You can also specify that mwm manage all screens by assigning a value of True to the multiScreen resource variable.

-name *app_name*

Specifies the name under which resources for the window manager should be found.

-screens *screen_name*[*screen_name*]...

Assigns resource names to the screens *mwm* is managing. (By default, the screen number is used as the *screen_name*.) If *mwm* is managing a single screen, only the first name in the list is used. If *mwm* is managing multiple screens, the names are assigned to the screens in order, starting with screen 0. If there are more screens

than names, resources for the remaining screens will be retrieved using the first *screen_name*.

-xrm *resourcestring*

Specifies a resource name and value to override any defaults. This option is very useful for setting resources that do not have explicit command-line arguments.

Window Manager Components

The *mwm* window frame contains various components that perform different functions. The title bar stretches across the top of the window and contains the title area and the minimize, maximize, and window menu buttons. The title area displays the window title and can be used to move the window. The minimize button iconifies the window, while the maximize button enlarges the window to fill the entire screen. The window menu button posts the **Window Menu**. The resize border handles surround the window; they are used to resize the window in a particular direction. A window can also have an optional matte decoration between the client area and the window frame. The matte is not part of the window frame and it has no functionality. At times, *mwm* uses dialog boxes or feedback windows to communicate with the user.

An icon is a small graphic representation of a window. When a window is iconified using the minimize button, it is replaced on the screen by its icon. Iconifying windows reduces clutter on the screen. *mwm* provides a separate window, call the icon box, that can hold icons. Using the icon box keeps icons from cluttering the screen.

By default, *mwm* uses an explicit keyboard selection policy, which means that once a window has the keyboard focus, it keeps it until another window is explicitly given the focus. Windows can overlap, which means that they are arranged in a global stacking order on the screen. A window that is higher in the stacking order obscures windows below it in the stacking order if they overlap. Each application has its own local stacking order; transient windows remain above their parents by default in the local stacking order.

Customization

Like any X application, *mwm* uses resources to control its appearance and behavior. The window manager builds its resource database just like any other X client. Mwm is the resource class name for *mwm*. You can place mwm resources in your regular resource file (*.Xdefaults*) in your home directory or you can create a file called *Mwm* (also in your home directory) for *mwm* resources only. If you place conflicting specifications in both files, the resources in*.Xdefaults* take precedence.

The default operation of the mouse, the keyboard, and menus in *mwm* is controlled by a system-wide resource description file, *system.mwmrc*. This file describes the contents of the **Window Menu** and **Root Menu**, as well as the key and button combinations that manage windows. To modify the behavior of *mwm*, you can edit a copy of this file in your home directory. The version of the file in your home directory should be called.*mwmrc*, unless you specify an alternate name using the configFile resource.

An *mwm* resource description file is a standard text file. Items are separated by blanks, tabs, and newlines. A line that begins with an exclamation mark (!*) o*r a number sign *(#*) is treated as a comment. If a line ends with a backslash (*)*, the subsequent line is considered a continuation of that line.

Component Appearance Resources

mwm provides some resources that specify the appearance of particular window manager components, such as the window frame, menus, and icons. Component appearance resources can be specified for particular window manager components or all components. To specify a resource for all components, use the following syntax:

Mwm**resource_name: resource_value*

The window manager components have the following resource names associated with them:

Component	ResourceName
Menu	menu
Icon	icon
Client window frame	client
Feedback/dialog box	feedback
Title bar	title

These resource names can be used to specify particular window manager components in a resource specification. To specify a resource for a specific component, use the following syntax:

Mwm*[*component_name*]**resource_name: resource_value*

The title bar is a descendant of the client window frame, so you can use title to specify the appearance of the title bar separately from the rest of the window frame. You can also specify resources for individual menus by using *menu*, followed by the name of the menu.

The following component appearance resources apply to all window manager components. Unless a default value is specified, the default varies based on system specifics such as the visual type of the screen:

background (class Background)
: Specifies the background color.

backgroundPixmap (class BackgroundPixmap)
: Specifies the background pixmap of the *mwm* decoration when the window does not have the input focus.

bottomShadowColor (class Foreground)
: Specifies the color to be used for the lower and right bevels of the window manager decoration.

bottomShadowPixmap (class BottomShadowPixmap)
: Specifies the pixmap to be used for the lower and right bevels of the window manager decoration.

fontList (class FontList)
: Specifies the font to be used in the window manager decoration. The default is fixed.

foreground (class Foreground)
: Specifies the foreground color.

saveUnder (class SaveUnder)
: Specifies whether save unders are used for *mwm* components. The default value is False, which means that save unders are not used on any window manager frames.

topShadowColor (class Background)
: Specifies the color to be used for the upper and left bevels of the window manager decoration.

topShadowPixmap (class TopShadowPixmap)
: Specifies the pixmap to be used for the upper and left bevels of the window manager decoration.

The following component appearance resources apply to the window frame and icons. Unless a default value is specified, the default varies based on system specifics such as the visual type of the screen:

activeBackground (class Background)
: Specifies the background color of the *mwm* decoration when the window has the input focus.

activeBackgroundPixmap (class ActiveBackgroundPixmap)
Specifies the background pixmap of the *mwm* decoration when the window has the input focus.

activeBottomShadowColor (class Foreground)
Specifies the bottom shadow color of the *mwm* decoration when the window has the input focus.

activeBottomShadowPixmap (class BottomShadowPixmap)
Specifies the bottom shadow pixmap of the *mwm* decoration when the window has the input focus.

activeForeground (class Foreground)
Specifies the foreground color of the *mwm* decoration when the window has the input focus.

activeTopShadowColor (class Background)
Specifies the top shadow color of the *mwm* decoration when the window has the input focus.

activeTopShadowPixmap (class TopShadowPixmap)
Specifies the top shadow Pixmap of the *mwm* decoration when the window has the input focus.

General Appearance and Behavior Resources

mwm also provides resources that control the appearance and behavior of the window manager as a whole. These resources specify features such as the focus policy, interactive window placement, and the icon box. To specify a general appearance and behavior resource, use the following syntax:

Mwm**resource_name: resource_value*

The following general appearance and behavior resources can be specified:

autoKeyFocus (class AutoKeyFocus)
If True (the default), when the focus window is withdrawn from window management or is iconified, the focus bounces back to the window that previously had the focus. This resource is available only when keyboardFocusPolicy is explicit. If False, the input focus is not set automatically. autoKeyFocus and startupKeyFocus should both be True to work properly with tear-off menus.

autoRaiseDelay (class AutoRaiseDelay)
Specifies the amount of time (in milliseconds) that *mwm* will wait before raising a window after it receives the input focus. The default is 500. This resource is available only when focusAutoRaise is True and the keyboardFocusPolicy is pointer.

bitmapDirectory (class BitmapDirectory)

Identifies the directory to be searched for bitmaps referenced by *mwm* resources (if an absolute pathname to the bitmap file is not given). The default is */usr/-include/-X11/-bitmaps*, which is considered the standard location on many systems. Note, however, that the location of the bitmap directory may vary in different environments. If a bitmap is not found in the specified directory, XBMLANGPATH is searched.

clientAutoPlace (class ClientAutoPlace)

Specifies the location of a window when the user has not specified a location. If True (the default), windows are positioned with the upper-left corners of the frames offset horizontally and vertically, so that no two windows completely overlap. If False, the currently configured position of the window is used. In either case, *mwm* attempts to place the windows totally on screen.

colormapFocusPolicy (class ColormapFocusPolicy)

Specifies the colormap focus policy. Takes three possible values: keyboard, pointer, and explicit. If keyboard (the default) is specified, the input focus window has the colormap focus. If explicit is specified, a colormap selection action is done on a client window to set the colormap focus to that window. If pointer is specified, the client window containing the pointer has the colormap focus.

configFile (class ConfigFile)

Specifies the pathname for the *mwm* startup file. The default startup file is.*mwmrc*.
mwm searches for the configuration file in the user's home directory. If the configFile resource is not specified or the file does not exist, *mwm* defaults to an implementation-specific standard directory (the default is */usr/lib/X11/system.mwmrc*).

If the LANG environment variable is set, *mwm* looks for the configuration file in a *$LANG* subdirectory first. For example, if the LANG environment variable is set to Fr (for French), *mwm* searches for the configuration file in the directory *$HOME/Fr* before it looks in *$HOME*. Similarly, if the configFile resource is not specified or the file does not exist, *mwm* defaults to */usr/lib/X11/$LANG/system.mwmrc* before it reads */usr/lib/X11/system.mwmrc*.

If the configFile pathname does not begin with ~/, *mwm* considers it to be relative to the current working directory.

deiconifyKeyFocus (class DeiconifyKeyFocus)

If True (the default), a window receives the input focus when it is normalized (deiconified). This resource applies only when the keyboardFocusPolicy is explicit.

doubleClickTime (class DoubleClickTime)

Specifies the maximum time (in milliseconds) between the two clicks of a double click. The default is the display's multi-click time.

enableWarp (class EnableWarp)

If True (the default), causes *mwm* to *warp* the pointer to the center of the selected window during resize and move operations invoked using keyboard accelerators. (The cursor symbol disappears from its current location and reappears at the center of the window.) If False, *mwm* leaves the pointer at its original place on the screen, unless the user explicitly moves it.

enforceKeyFocus (class EnforceKeyFocus)

If True (the default), the input focus is always explicitly set to selected windows even if there is an indication that they are "globally active" input windows. (An example of a globally active window is a scrollbar that can be operated without setting the focus to that client.) If the resource explicitly set to globally active windows.

iconAutoPlace (class IconAutoPlace)

Specifies whether the window manager arranges icons in a particular area of the screen or places each icon where the window was when it was iconified. If True (the default), icons are arranged in a particular area of the screen, determined by the iconPlacement resource. If False, an icon is placed at the location of the window when it is iconified.

iconClick (class IconClick)

If True (the default), the **Window Menu** is displayed when the pointer is clicked on an icon.

interactivePlacement (class InteractivePlacement)

If True, specifies that new windows are to be placed interactively on the screen using the pointer. When a client is run, the pointer shape changes to an upper-left corner cursor; move the pointer to the location you want the window to appear and click the first button; the window is displayed in the selected location. If False (the

default), windows are placed according to the initial window configuration attributes.

keyboardFocusPolicy (class KeyboardFocusPolicy)
: If explicit focus is specified (the default), placing the pointer on a window (including the frame) or icon and pressing the first pointer button focuses keyboard input on the client. If pointer is specified, the keyboard input focus is directed to the client window on which the pointer rests (the pointer can also rest on the frame).

lowerOnIconify (class LowerOnIconify)
: If True (the default), a window's icon is placed on the bottom of the stack when the window is iconified. If False, the icon is placed in the stacking order at the same place as its associated window.

moveThreshold (class MoveThreshold)
: Controls the sensitivity of dragging operations, such as those used to move windows and icons on the display. Takes a value of the number of pixels that the pointing device is moved while a button is held down before the move operation is initiated. The default is 4. This resource helps prevent a window or icon from moving when you click or double click and inadvertently jostle the pointer while a button is down.

multiScreen (class MultiScreen)
: If False (the default), *mwm* manages only a single screen. If True, *mwm* manages all screens on the display.

passButtons (class PassButtons)
: Specifies whether button press events are passed to clients after the events are used to invoke a window manager function in the client context. If False (the default), button presses are not passed to the client. If True, button presses are passed to the client. The window manager function is done in either case.

passSelectButton (class PassSelectButton)
: Specifies whether select button press events are passed to clients after the events are used to invoke a window manager function in the client context. If True (the default), button presses are passed to the client window. If False, button presses are not passed to the client. The window manager function is done in either case.

positionIsFrame (class PositionIsFrame)
: Specifies how *mwm* should interpret window position information from the WM_NORMAL_HINTS property and from configuration

requests. If True (the default), the information is interpreted as the position of the *mwm* client window frame. If False, it is interpreted as being the position of the client area of the window.

positionOnScreen (class PositionOnScreen)

If True (the default), specifies that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen. If a window is larger than the size of the screen, at least the upper-left corner of the window is placed is on the screen. If False, windows are placed in the requested position even if totally off the screen.

quitTimeout (class QuitTimeout)

Specifies the amount of time (in milliseconds) that *mwm* will wait for a client to update the WM_COMMAND property after *mwm* has sent the WM_SAVE_-YOURSELF message. The default is 1000. (See the f.kill function for additional information.)

raiseKeyFocus (class RaiseKeyFocus)

If True, specifies that a window raised by means of the f.normalize_-and_raise function also receives the input focus. This function is available only when the keyboardFocusPolicy is explicit. The default is False.

screens (class Screens)

Assigns resource names to the screens *mwm* is managing. If *mwm* is managing a single screen, only the first name in the list is used. If *mwm* is managing multiple screens, the names are assigned to the screens in order, starting with screen 0.

showFeedback (class ShowFeedback)

Specifies whether *mwm* feedback windows and confirmation dialog boxes are displayed. (Feedback windows are used to display: window coordinates during interactive placement and subsequent moves; and dimensions during resize operations. A typical confirmation dialog is the window displayed to allow the user to allow or cancel a window manager restart operation.)

showFeedback accepts a list of options, each of which corresponds to the type of feedback given in a particular circumstance. Depending on the syntax in which the options are entered, you can either enable or disable a feedback option (as explained later).

The possible feedback options are: all, which specifies that *mwm* show all types of feedback (this is the default); behavior, which

specifies that feedback is displayed to confirm a behavior switch; kill, which specifies that feedback is displayed on receipt of a KILL signal; move, which specifies that a box containing the coordinates of a window or icon is displayed during a move operation; placement, which specifies that a box containing the position and size of a window is displayed during initial (interactive) placement; quit, which specifies that a dialog box is displayed so that the user can confirm (or cancel) the procedure to quit *mwm*; resize, which specifies that a box containing the window size is displayed during a resize operation; restart, which displays a dialog box so that the user can confirm (or cancel) an *mwm* restart procedure; the none option specifies that no feedback is shown.

To limit feedback to particular cases, you can use one of two syntaxes: with the first syntax, you disable feedback in specified cases (all other default feedback is still used); with the second syntax, you enable feedback only in specified cases. You supply this resource with a list of options to be enabled or disabled. If the first item is preceded by a minus sign, feedback is disabled for all options in the list. If the first item is preceded by a plus sign (or no sign is used), feedback is enabled only for options in the list.

startupKeyFocus (class StartupKeyFocus)
: If True (the default), the input focus is transferred to a window when the window is mapped (i.e., initially managed by the window manager). This function is available only when keyboardFocusPolicy is explicit. startupKeyFocus and autoKeyFocus should both be True to work properly with tear-off menus.

wMenuButtonClick (class WMenuButtonClick)
: If True (the default), a pointer button click on the window menu button displays the **Window Menu** and leaves it displayed.

wMenuButtonClick2 (class WMenuButtonClick2)
: If True, double clicking on the window menu button removes the client window, which means that f.kill is invoked.

Screen-Specific Resources

Some *mwm* resources can be applied on a per-screen basis. To specify a screen-specific resource, use the following syntax:

Mwm**screen_number*resource_name: resource_value*

Screen-specific specifications take precedence over specifications for all screens. Screen-specific resources can be specified for all screens using the following syntax:

Mwm**resource_name: resource_value*

buttonBindings (class ButtonBindings)
: Identifies the set of button bindings to be used for window management functions; must correspond to a set of button bindings specified in the *mwm* startup file. Button bindings specified in the startup file are merged with built-in default bindings. The default is DefaultButtonBindings.

cleanText (class CleanText)
: Specifies whether text that appears in *mwm* title and feedback windows is displayed over the existing background pattern. If True (the default), text is drawn with a clear (no stipple) background. (Only the stippling in the area immediately around the text is cleared.) This enhances readability, especially on monochrome systems where a backgroundPixmap is specified. If False, text is drawn on top of the existing background.

fadeNormalIcon (class FadeNormalIcon)
: If True, an icon is greyed out when it has been normalized. The default is False.

feedbackGeometry (class FeedbackGeometry)
: Specifies the position of the small, rectangular feedback box that displays coordinate and size information during move and resize operations. By default, the feedback window appears in the center of the screen. This resource takes the argument:

  [=]±*xoffset*±*yoffset*

  With the exception of the optional leading equal sign, this string is identical to the second portion of the standard geometry string. Note that feedbackGeometry allows you to specify location only. The size of the feedback window is not configurable using this resource. Available as of *mwm* version 1.2 and later.

frameBorderWidth (class FrameBorderWidth)
: Specifies the width in pixels of a window frame border, without resize handles. (The border width includes the three-dimensional shadows.) The default is determined according to screen specifics.

frameStyle

In Motif 2.0 and later, specifies the frame appearance of decoration windows and borders: the value WmRECESSED makes the window appear recessed into the border, the value WmSLAB gives a flat window and border.

iconBoxGeometry (class IconBoxGeometry)

Specifies the initial position and size of the icon box. Takes as its argument the standard geometry string:

*width*x*height*±*xoff*±*yoff*

where *width* and *height* are measured in icons. The default geometry string is 6x1+0-0, which places an icon box six icons wide by one icon high in the lower-left corner of the screen.

You can omit either the dimensions or the x and y offsets from the geometry string and the defaults apply. If the offsets are not provided, the iconPlacement resource is used to determine the initial placement.

The actual screen size of the icon box depends on the iconImageMaximum and iconDecoration resources, which specify icon size and padding. The default value for size is (6 × *icon_width* + padding) wide by (1 × *icon_height* + padding) high.

iconBoxName (class IconBoxName)

Specifies the name under which icon box resources are to be found. The default is iconbox.

iconBoxSBDisplayPolicy (class IconBoxSBDisplayPolicy)

Specifies what scrollbars are displayed in the icon box. The resource has three possible values: all, vertical, and horizontal. If all is specified (the default), both vertical and horizontal scrollbars are displayed at all times. vertical specifies that a single vertical scrollbar is displayed and sets the orientation of the icon box to horizontal, regardless of the iconBoxGeometry specification. horizontal specifies that a single horizontal scrollbar is displayed in the icon box and sets the orientation of the icon box to vertical, regardless of the iconBoxGeometry specification.

iconBoxTitle (class IconBoxTitle)

Specifies the name to be used in the title area of the icon box. The default is Icons.

iconDecoration (class IconDecoration)

Specifies how much icon decoration is used. The resource value takes four possible values (multiple values can also be supplied): label, which specifies that only the label is displayed; image, which specifies that only the image is displayed; and activelabel, which specifies that a label (not truncated to the width of the icon) is used when the icon has the focus.

The default decoration for icons in an icon box is label image, which specifies that both the label and image parts are displayed. The default decoration for individual icons on the screen proper is activelabel label image.

iconImageMaximum (class IconImageMaximum)

Specifies the maximum size of the icon image. Takes a value of *widthxheight* (e.g., 80×80). The maximum size supported is 128×128. The default is 50×50.

iconImageMinimum (class IconImageMinimum)

Specifies the minimum size of the icon image. Takes a value of *widthxheight* (e.g., 36×48). The minimum size supported is 16×16 (which is also the default).

iconPlacement (class IconPlacement)

Specifies an icon placement scheme. Note that this resource is only useful when useIconBox is False (the default). The iconPlacement resource takes a value of the syntax:

primary_layout secondary_layout [tight]

There are four possible layout policies. top specifies that icons are placed from the top of the screen to the bottom, bottom specifies a bottom-to-top arrangement, left specifies that icons are placed from the left to the right, and right specifies a right-to-left arrangement. The optional argument tight specifies that there is no space between icons.

The *primary_layout* specifies whether icons are placed in a row or a column and the direction of placement. The *secondary_layout* specifies where to place new rows or columns. For example, a value of top right specifies that icons should be placed from top to bottom on the screen and that columns should be added from right to left on the screen.

A horizontal (vertical) layout value should not be used for both the *primary_layout* and the *secondary_layout*. For example, do not use top for the *primary_layout* and bottom for the *secondary_layout*.

The default placement is left bottom (i.e., icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows are added from the bottom of the screen to the top of the screen).

iconPlacementMargin (class IconPlacementMargin)

Sets the distance from the edge of the screen at which icons are placed. The value should be greater than or equal to 0. A default value is used if an invalid distance is specified. The default value is equal to the space between icons as they are placed on the screen, which is based on maximizing the number of icons in each row and column.

keyBindings (class KeyBindings)

Identifies the set of key bindings to be used for window management functions; must correspond to a set of key bindings specified in the *mwm* startup file. Note that key bindings specified in the startup file replace the built-in default bindings. The default is DefaultKeyBindings.

limitResize (class LimitResize)

If True (the default), the user is not allowed to resize a window to greater than the maximum size.

maximumMaximumSize (class MaximumMaximumSize)

Specifies the maximum size of a client window (as set by the user or client). Takes a value of *widthxheight* (e.g., 1024x1024) where *width* and *height* are in pixels. The default is twice the screen width and height.

moveOpaque (class MoveOpaque)

If False (the default), when you move a window or icon, its outline is moved before it is redrawn in the new location. If True, the actual (and thus, opaque) window or icon is moved. Available as of *mwm* version 1.2 and later.

resizeBorderWidth (class ResizeBorderWidth)

Specifies the width in pixels of a window frame border, with resize handles. (The border width includes the three-dimensional shadows.) The default is determined according to screen specifics.

resizeCursors (class ResizeCursors)

If True (the default), the resize cursors are always displayed when the pointer is in the window resize border.

transientDecoration (class TransientDecoration)

Specifies the amount of decoration *mwm* puts on transient windows. The decoration specification is exactly the same as for the clientDecoration (client-specific) resource. Transient windows are identified by the WM_TRANSIENT_FOR property, which is added by the client to indicate a relatively temporary window. The default is menu title, which specifies that transient windows have resize borders and a title bar with a window menu button. If the client application also specifies which decorations the window manager should provide, *mwm* uses only those features that both the client and the transientDecoration resource specify.

transientFunctions (class TransientFunctions)

Specifies which window management functions are applicable (or not applicable) to transient windows. The function specification is exactly the same as for the clientFunctions (client-specific) resource. The default is -minimize maximize. If the client application also specifies which window management functions should be applicable, *mwm* provides only those functions that both the client and the transientFunctions resource specify.

useIconBox (class UseIconBox)

If True, icons are placed in an icon box. By default, the individual icons are placed on the root window.

Client-Specific Resources

Some *mwm* resources can be set to apply to certain client applications or classes of applications. To specify a client-specific resource, use the following syntax:

Mwm**client_name*resource_name: resource_value*

Client-specific specifications take precedence over specifications for all clients. Client- specific resources can be specified for all clients using the following syntax:

Mwm**resource_name: resource_value*

The class name defaults can be used to specify resources for clients that have an unknown name and class.

The following client-specific resources can be specified:

clientDecoration (class ClientDecoration)

Specifies the amount of window frame decoration. The default frame is composed of several component parts: the title bar, resize handles, border, and the minimize, maximize, and window menu buttons. You can limit the frame decoration for a client using the clientDecoration resource.

clientDecoration accepts a list of options, each of which corresponds to a part of the client frame. The options are: maximize, minimize, menu, border, title, resize, all, which encompasses all decorations previously listed, and none, which specifies that no decorations are used.

Some decorations require the presence of others; if you specify such a decoration, any decorations required with it are used automatically. Specifically, if any of the command buttons are specified, a title bar is also used; if resize handles or a title bar is specified, a border is also used.

By default, a client window has all decoration. To specify only certain parts of the default frame, you can use one of two syntaxes: with the first syntax, you disable certain frame features; with the second syntax, you enable only certain features. You supply clientDecoration with a list of options to be enabled or disabled. If the first item is preceded by a minus sign, the features in the list are disabled. If the first item is preceded by a plus sign (or no sign is used), only those features listed are enabled.

clientFunctions (class ClientFunctions)

Specifies whether certain *mwm* functions can be invoked on a client window. The only functions that can be controlled are those that are executable using the pointer on the default window frame. clientFunctions accepts a list of options, each of which corresponds to an *mwm* function. The options are: resize, move, minimize, maximize, close, all, which encompasses all of the previously listed functions, and none, which specifies that no default functions are allowed.

By default, a client recognizes all functions. To limit the functions a client recognizes, you can use one of two syntaxes: with the first syntax, you disallow certain functions; with the second syntax, you allow only certain functions. You supply clientFunctions with a list of options (corresponding to functions) to be allowed or disallowed. If the first item is preceded by a minus sign, the functions in the list are disallowed. If the first option is preceded by a plus sign (or no

sign is used), only those functions listed are allowed.
A less than obvious repercussion of disallowing a particular function is that the client window frame is also altered to prevent your invoking that function. For instance, if you disallow the f.resize function for a client, the client's frame does not include resize borders. In addition, the **Size** item on the **Window Menu**, which invokes the f.resize function, no longer appears on the menu.

If the client application also specifies which window management functions should be applicable, *mwm* provides only those functions that both the client and the clientFunctions resource specify.

focusAutoRaise (class FocusAutoRaise)
: If True, a window is raised when it receives the input focus. Otherwise, directing focus to a window does not affect the stacking order. The default depends on the value assigned to the keyboardFocusPolicy resource. If the keyboardFocusPolicy is explicit, the default for focusAutoRaise is True. If the keyboardFocusPolicy is pointer, the default for focusAutoRaise is False.

iconImage (class IconImage)
: Specifies the pathname of a bitmap file to be used as an icon image for a client. The default is to display an icon image supplied by the window manager. If the useClientIcon resource is set to True, an icon image supplied by the client takes precedence over an icon image supplied by the user.

iconImageBackground (class Background)
: Specifies the background color of the icon image. The default is the color specified by Mwm*background or Mwm*icon*background.

iconImageBottomShadowColor (class Foreground)
: Specifies the bottom shadow color of the icon image. The default is the color specified by Mwm*icon*bottomShadowColor.

iconImageBottomShadowPixmap (class BottomShadowPixmap)
: Specifies the bottom shadow pixmap of the icon image. The default is the pixmap specified by Mwm*icon*bottomShadowPixmap.

iconImageForeground (class Foreground)
: Specifies the foreground color of the icon image. The default varies based on the icon background.

iconImageTopShadowColor (class Background)
: Specifies the top shadow color of the icon image. The default is the color specified by Mwm*icon*topShadowColor.

iconImageTopShadowPixmap (class TopShadowPixmap)
: Specifies the top shadow Pixmap of the icon image. The default is the pixmap specified by Mwm*icon*topShadowPixmap.

matteBackground (class Background)
: Specifies the background color of the matte. The default is the color specified by Mwm*background or Mwm*client*background. This resource is only relevant if matteWidth is positive.

matteBottomShadowColor (class Foreground)
: Specifies the bottom shadow color of the matte. The default is the color specified by Mwm*bottomShadowColor or Mwm*client*bottomShadowColor. This resource is only relevant if matteWidth is positive.

matteBottomShadowPixmap (class BottomShadowPixmap)
: Specifies the bottom shadow pixmap of the matte. The default is the pixmap specified by Mwm*bottomShadowPixmap or Mwm*client*bottomShadowPixmap. This resource is only relevant if matteWidth is positive.

matteForeground (class Foreground)
: Specifies the foreground color of the matte. The default is the color specified by Mwm*foreground or Mwm*client*foreground. This resource is only relevant if matteWidth is positive.

matteTopShadowColor (class Background)
: Specifies the top shadow color of the matte. The default is the color specified by Mwm*topShadowColor or Mwm*client*topShadowColor. This resource is only relevant if matteWidth is positive.

matteTopShadowPixmap (class TopShadowPixmap)
: Specifies the top shadow pixmap of the matte. The default is the pixmap specified by Mwm*topShadowPixmap or Mwm*client*topShadowPixmap. This resource is only relevant if matteWidth is positive.

matteWidth (class MatteWidth)
: Specifies the width of the matte. The default is 0, which means no matte is used.

maximumClientSize (class MaximumClientSize)

Specifies how a window is to be maximized, either to a specific size (*widthxheight*), or as much as possible in a certain direction (vertical or horizontal). If the value is of the form *widthxheight*, the width and height are interpreted in the units used by the client. For example, *xterm* measures width and height in font characters and lines.

If maximumClientSize is not specified, and the WM_NORMAL_HINTS property is set, the default is obtained from it. If WM_NORMAL_HINTS is not set, the default is the size (including borders) that fills the screen. *mwm* also uses maximumMaximumSize to constrain the value in this case.

useClientIcon (class UseClientIcon)
If True, an icon image supplied by the client takes precedence over an icon image supplied by the user. The default is False.

usePPosition (class UsePPosition)
Specifies whether *mwm* uses initial coordinates supplied by the client application. If True, *mwm* always uses the program specified position. If False, *mwm* never uses the program specified position. The default is nonzero, which means that *mwm* will use any program specified position except 0,0. Available as of *mwm* version 1.2 and later.

windowMenu (class WindowMenu)
Specifies a name for the **Window Menu** (which must be defined in the startup file). The default is DefaultWindowMenu.

Functions

mwm supports a number of functions that can be bound to different key and button combinations and assigned to menus in the *mwm* resource description file (*system.mwmrc* or *.mwmrc*). Most window manager functions can be used in key bindings, button bindings, and menus. The function descriptions below note any exceptions to this policy. Most window manager functions can also be specified for three contexts: root, window, and icon. The root context means that the function is applied to the root window, window means that the function is applied to the selected client window, and icon means that the function is applied to the selected icon. The function descriptions below note any functions that cannot be used in all three contexts.

When a function is specified with the context icon | window and you invoke the function from the icon box, the function applies to the icon box itself, rather than to any of the icons it contains.

A function is treated as f.nop if it is not a valid function name, if it is specified inappropriately, or if it is invoked in an invalid way.

mwm recognizes the following functions:

f.beep
: Causes a beep from the keyboard.

f.circle_down [icon | window]
: Causes the window or icon on the top of the stack to be lowered to the bottom of the stack. If the icon argument is specified, the function applies only to icons. If the window argument is specified, the function applies only to windows.

f.circle_up [icon | window]
: Causes the window or icon on the bottom of the stack to be raised to the top. If the icon argument is specified, the function applies only to icons. If the window argument is specified, the function applies only to windows.

f.exec[*command*]
![*command*]
: Executes *command* using the shell specified by the MWMSHELL environment variable. If MWMSHELL is not set, the command is executed using the shell specified by the SHELL environment variable; otherwise, the command is executed using */bin/sh*.

f.focus_color
: Sets the colormap focus to a client window. If this function is invoked in the root context, the default colormap (specified by X for the screen where *mwm* is running) is installed and there is no specific client window colormap focus. For the f.focus_color function to work, the colormapFocusPolicy should be specified as explicit; otherwise the function is treated as f.nop.

f.focus_key
: Sets the input focus to a window or icon. For the f.focus_key function to work, the keyboardFocusPolicy should be specified as explicit. If keyboardFocusPolicy is not explicit or if the function is invoked in the root context, it is treated as f.nop.

f.kill
: Terminates a client. It sends the WM_DELETE_WINDOW message to the selected window if the client application has requested it through the WM_PROTOCOLS property. The application is supposed to respond to the message by removing the indicated win-

dow. If the WM_SAVE_YOURSELF protocol is set up and the WM_DELETE_WINDOW protocol is not, the client is sent a message that indicates that the client needs to prepare to be terminated. If the client does not have the WM_DELETE_WINDOW or WM_SAVE_YOURSELF protocol set, the f.kill function causes a client's X connection to be terminated.

f.lower [-*client* | within | *freeFamily*]

Without arguments, lowers a window or icon to the bottom of the stack. By default, the context in which the function is invoked indicates to the window or icon to lower. If an application window has one or more transient windows (e.g., dialog boxes), the transient windows are lowered with the parent (within the global stack) and remain on top of it. If the *-client* argument is specified, the function is invoked on the named client. client must be the instance or class name of a program. The within argument is used to lower a transient window within the application's local window hierarchy; all transients remain above the parent window and that window remains in the same position in the global window stack. In practice, this function is only useful when there are two or more transient windows and you want to shuffle them. The freeFamily argument is used to lower a transient below its parent in the application's local window hierarchy. Again, the parent is not moved in the global window stack. However, if you use this function on the parent, the entire family stack is lowered within the global stack.

f.maximize

Causes a window to be redisplayed at its maximum size. This function cannot be invoked in the context root or on a window that is already maximized.

f.menu *menu_name*

Associates a cascading menu with a menu item or associates a menu with a button or key binding. The *menu_name* argument specifies the menu.

f.minimize

Causes a window to be minimized (i.e., iconified). When no icon box is being used, icons are placed on the bottom of the stack, which is generally in the lower-left corner of the screen. If an icon box is being used, icons are placed inside the box. This function cannot be invoked in the context root or on an iconified window.

f.move
: Allows you to move a window interactively, using the pointer.

f.next_cmap
: Installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [icon | window | transient]
: Without any arguments, this function advances the input focus to the next window or icon in the stack. You can specify icon or window to make the function apply only to icons or windows, respectively. Generally, the focus is moved to windows that do not have an associated secondary window that is application modal. If the transient argument is specified, transient windows are also traversed. Otherwise, if only window is specified, focus is moved to the last window in a transient group to have the focus. For this function to work, keyboardFocusPolicy must be explicit; otherwise, the function is treated as f.nop.

f.nop
: Specifies no operation.

f.normalize
: Causes a client window to be displayed at its normal size. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.normalize_and_raise
: Causes the client window to be displayed at its normal size and raised to the top of the stack. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.pack_icons
: Rearranges icons in an optimal fashion based on the layout policy being used, either on the root window or in the icon box.

f.pass_keys
: Toggles processing of key bindings for window manager functions. When key binding processing is disabled, all keys are passed to the window with the keyboard input focus and no window manager functions are invoked. If the f.pass_keys function is set up to be invoked with a key binding, the binding can be used to toggle key binding processing.

f.post_wmenu

Displays the **Window Menu**. If a key is used to display the menu and a window menu button is present, the upper-left corner of the menu is placed at the lower-left corner of the command button. If no window menu button is present, the menu is placed in the upper-left corner of the window.

f.prev_cmap

This function installs the previous colormap in the list of colormaps for the window with the colormap focus.

f.prev_key [*icon* | *window* | *transient*]

Without any arguments, this function moves the input focus to the previous window or icon in the stack. You can specify *icon* or *window* to make the function apply only to icons or windows, respectively. Generally, the focus is moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, transient windows are also traversed. Otherwise, if only *window* is specified, focus is moved to the last window in a transient group to have the focus. For this function to work, keyboardFocusPolicy must be explicit; otherwise, the function is treated as f.nop.

f.quit_mwm

Stops the *mwm* window manager. Note that this function does not stop the X server. This function cannot be invoked from a non-root menu.

f.raise *[-client* | *within* | *freeFamily*]

Without arguments, raises a window or icon to the top of the stack. By default, the context in which the function is invoked indicates the window or icon to raise. If an application window has one or more transient windows (e.g., dialog boxes), the transient windows are raised with the parent (within the global stack) and remain on top of it. If the *-client* argument is specified, the function is invoked on the named client. client must be the instance or class name of a program. The *within* argument is used to raise a transient window within the application's local window hierarchy; all transients remain above the parent window and that window remains in the same position in the global window stack. In practice, this function is only useful when there are two or more transient windows and you want to shuffle them.

The *freeFamily* argument raises a transient to the top of the application's local window hierarchy. The parent window is also raised to the top of the global stack.

f.raise_lower [*within* | *freeFamily*]
: Raises a primary application window to the top of the stack or lowers a window to the bottom of the stack, as appropriate to the context. The within argument is intended to raise a transient window within the application's local window hierarchy. All transients remain above the parent window and the parent window should also remain in the same position in the global window stack. If the transient is not obscured by another window in the local stack, the transient window is lowered within the family. The preceding paragraph describes how within *should* work. However, we have found that the parent window does not always remain in the same position in the global window stack. The freeFamily argument raises a transient to the top of the family stack and also raises the parent window to the top of the global stack. If the transient is not obscured by another window, this function lowers the transient to the bottom of the family stack and lowers the family in the global stack.

f.refresh
: Redraws all windows.

f.refresh_win
: Redraws a single window.

f.resize
: Allows you to resize a window interactively, using the pointer.

f.restart
: Restarts the *mwm* window manager. The function causes the current *mwm* process to be stopped and a new *mwm* process to be started. It cannot be invoked from a non-root menu.

f.restore
: Causes the client window to be displayed at its previous size. If invoked on an icon, f.restore causes the icon to be converted back to a window at its previous size. Thus, if the window was maximized, it is restored to this state. If the window was previously at its normal size, it is restored to this state. If invoked on a maximized window, the window is restored to its normal size. This function

cannot be invoked in the context root or on a window that is already at its normal size.

f.restore_and_raise
: Causes the client window to be displayed at its previous size and raised to the top of the stack. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.screen [*next* | *prev* | *back* | *screen_number*]
: Causes the pointer to be warped to another screen, which is determined by one of four mutually exclusive parameters. The *next* argument means skip to the next managed screen, *prev* means skip back to the previous managed screen, back means skip to the last screen visited, and *screen_number* specifies a particular screen. Screens are numbered beginning at 0.

f.send_msg *message_number*
: Sends a message of the type _MOTIF_WM_MESSAGES to a client; the message type is indicated by the *message_number* argument. The message is sent only if the client's _MOTIF_WM_MESSAGES property includes *message_number.* If a menu item is set up to invoke f.send_msg and the *message_number* is not included in the client's _MOTIF_WM_MESSAGES property, the menu item label is greyed out, which indicates that it is not available for selection.

f.separator
: Creates a divider line in a menu. Any associated label is ignored.

f.set_behavior
: Restarts *mwm*, toggling between the default behavior for the particular system and the user's custom environment. In any case, a dialog box asks the user to confirm or cancel the action. By default this function is invoked using the following key sequence: Shift Ctrl Meta !.

f.title
: Specifies the title of a menu. The title string is separated from the menu items by a double divider line.

Event Specification

In order to specify button bindings, key bindings, and menu accelerators, you need to be able to specify events in the *mwm* resource description file. Use the following syntax to specify button events for button bindings:

[*modifier_key...*]<*button_event*>

The acceptable values for *modifier_key* are: Ctrl, Shift, Alt, Meta, Lock, Mod1, Mod2, Mod3, Mod4, and Mod5. *mwm* considers Alt and Meta to be equivalent.

The acceptable values for *button_event* are:

Btn1Down	Btn2Down	Btn3Down	Btn4Down	Btn5Down
Btn1Up	Btn2Up	Btn3Up	Btn4Up	Btn5Up
Btn1Click	Btn2Click	Btn3Click	Btn4Click	Btn5Click
Btn1Click2	Btn2Click2	Btn3Click2	Btn4Click2	Btn5Click2

Use the following syntax to specify key events for key bindings and menu accelerators:

[*modifier_key...*]<Key>*key_name*

Any X11 keysym name is an acceptable value for *key_name*.

Button Bindings

The buttonBindings resource specifies the name of a set of button bindings that control mouse behavior in mwm. You can create your own set of button bindings or use one of the sets defined in *system.mwmrc:* DefaultButtonBindings, ExplicitButtonBindings, or PointerButtonBindings. Use the following syntax to specify a set of button bindings:

```
Buttons button_set_name
{
    button context function
    button context function
    ...
    button context function
}
```

The *context* specifies where the pointer must be located for the button binding to work. The context is also used for window manager functions that are context-sensitive. The valid contexts for button bindings are root, window, icon, title, border, frame, and app. The title context refers to the title area of the frame. border refers to the frame exclusive of the title bar. frame refers to the entire frame. The app context refers to the application window proper. The window context includes the application window and the frame. A context specification can

include multiple contexts; use a vertical bar (|) to separate multiple context values.

Key Bindings

The keyBindings resource specifies the name of a set of key bindings that control keyboard behavior in *mwm*. You can create your own set of key bindings or use the default key bindings, DefaultKeyBindings, defined in *system.mwmrc*. Use the following syntax to specify a set of key bindings:

```
Keys key_set_name
{
    key context function
    key context function
    ...
    key context function
}
```

The *context* specifies where the keyboard focus must be for the key binding to work. The context is also used for window manager functions that are context-sensitive. The valid contexts for key bindings are root, window, icon, title, border, frame, and app. The title, border, frame, and app contexts are all equivalent to window. A context specification can include multiple contexts; use a vertical bar (|) to separate multiple context values.

Menus

The window manager functions f.post_wmenu and f.menu post menus. These functions both take the name of a menu to post. You can create your own menus or use the default menus defined in *system.mwmrc*: DefaultRootMenu and DefaultWindowMenu. Use the following syntax to specify a menu:

```
Menu menu_name
{
    label [mnemonic] [accelerator] function
    label [mnemonic] [accelerator] function
    ...
    label [mnemonic] [accelerator] function
}
```

Each line in a menu specification indicates the label for the menu item, optional keyboard mnemonics and accelerators, and the window manager function that is performed. *label* can be a string or a bitmap file. If the string contains multiple

words, it must be enclosed in quotation marks. A bitmap file specification is preceded by an at sign (*@*). A mnemonic is specified as _character. An accelerator specification uses the key event specification syntax.

The context of a window manager function that is activated from a menu is based on how the menu is posted. If it is posted from a button binding, the context of the menu is the context of the button binding. If it is posted from a key binding, the context of the menu is based on the location of the keyboard focus.

Environment

mwm uses the following environment variables:

HOME
: The user's home directory.

LANG
: The language to be used for the *mwm* message catalog and the *mwm* startup file.

XBMLANGPATH
: Used to search for bitmap files.

XFILESEARCHPATH
: Used to determine the location of system-wide class resource files. If the LANG variable is set, the *$LANG* subdirectory is also searched.

XUSERFILESEARCHPATH, XAPPLRESDIR
: Used to determine the location of user-specific class resource files. If the LANG variable is set, the *$LANG* subdirectory is also searched.

MWMSHELL, SHELL
: MWMSHELL specifies the shell to use when executing a command supplied as an argument to the f.exec function. If MWMSHELL is not set, SHELL is used.

Files

/usr/lib/X11/$LANG/system.mwmrc
/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/Mwm
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
$HOME/.motifbind

See Also

`XmIsMotifWMRunning`(1), `XmInstallImage`(1), `VendorShell`(2), `xmbind`(3)

Name

uil – the User Interface Language (UIL) compiler.

Syntax

uil [*options*] *file*

Description

The *uil* command invokes the User Interface Language (UIL) compiler. If the file does not contain any errors, the compiler generates a User Interface Description (UID) file that contains a compiled form of the input file. UIL is a specification language that can be used to describe the initial state of a user interface that uses the OSF/Motif widget set, as well as other widgets. The user interface for an application is created at run-time using the Motif Resource Manager (Mrm) library; the interface is based on compiled interface descriptions stored in one or more UID files.

Options

-I*pathname*

Specifies a search path for include files. By default, the current directory and */usr/include* are searched. Path names may be relative or absolute. The paths specified with this option are searched in order after the current directory and before */usr/include*.

-m

When specified with the -v option, the UIL compiler includes machine code in the listing file. The machine code provides binary and text descriptions of the data that is stored in the UID file. This option is useful for determining exactly how the compiler interprets a particular statement and how much storage is used for the variables, declarations, and assignments.

-o *ofile*

Specifies the name of the UID file to output. The default filename is *a.uid*. The customary suffix for UID files is *.uid*.

-s

Specifies that the UIL compiler set the locale before compiling any files. Setting the locale determines the behavior of locale-dependent routines like character string operations. Although setting the locale is an implementation-dependent operation, on ANSI-C-based systems, the locale is set with the call:

setlocale (LC_ALL, "")

See the setlocale() man page on your system for more information.

-v *lfile*

Directs the UIL compiler to produce a listing of the compilation. The file indicates the name of the output file. If this option is not specified, the compiler does not generate a listing. On UNIX systems, a filename of */dev/tty* usually causes the listing to be output on the terminal where *uil* was invoked.

-w

Directs the compiler to suppress warning and informational messages and to print only error messages. The default behavior is to print error, warning and informational messages.

-wmd *wfile*

Specifies a compiled Widget Meta-Language (WML) description file that is loaded in place of the default WML description. The default WML description file contains a description of all of the Motif widgets. This option is normally used to debug a WML description file without rebuilding the UIL compiler.

Environment

The LANG environment variable affects the way that the UIL compiler parses and generates compound strings, fonts, font sets, and font tables (font lists). The exact effect is described by the UIL reference pages for these types.

Example

```
% uil -o myfile.uid -v /dev/tty myfile.uil
% uil -I/project/include/uil -o mainui.uid mainui.uil
```

Bugs

If the LANG environment variable is set to an invalid value and the -s option is specified, the UIL compiler crashes.

See Also

`Uil`*(7)*.

Name

xmbind – configure virtual key bindings.

Syntax

xmbind [*options*] [*file*]

Availability

Motif 1.2 and later.

Description

The *xmbind* command configures the virtual key bindings for Motif applications. Since this action is performed by *mwm* on startup, *xmbind* is only needed when *mwm* is not being used or when a user wants to change the bindings without restarting *mwm*.

When a file is specified, its contents are used for the virtual bindings. Otherwise, *xmbind* uses the *.motifbind* file in the user's home directory. A sample specification is shown below:

```
osfBackSpace:   <Key>BackSpace
osfInsert:      <Key>InsertChar
osfDelete:      <Key>DeleteChar
```

If *xmbind* cannot find the *.motifbind* file, it loads the default virtual bindings for the server. *xmbind* searches for a vendor-specific set of bindings located using the file *xmbind.alias*. If this file exists in the user's home directory, it is searched for a pathname associated with the vendor string or the vendor string and vendor release. If the search is unsuccessful, Motif continues looking for *xmbind.alias* in the directory specified by XMBINDDIR or in */usr/lib/Xm/bindings* if the variable is not set. If this file exists, it is searched for a pathname as before. If either search locates a pathname and the file exists, the bindings in that file are used. An *xmbind.alias* file contains lines of the following form:

"*vendor_string*[*vendor_release*]"*bindings_file*

If *xmbind* still has not located any bindings, it loads fixed fallback default bindings.

The Motif toolkit uses a mechanism called *virtual bindings* to map one set of keysyms to another set. This mapping permits widgets and applications to use one set of keysyms in translation tables; applications and users can then customize the keysyms used in the translations based on the particular keyboard that is being used. Keysyms that can be used in this way are called *osf keysyms*. Motif maintains a mapping between the osf keysyms and the actual keysyms that represent keys on a particular keyboard. See the Introduction to Section 2, Motif and Xt Widget Classes, for more information about virtual bindings.

Options

-display[*host*]:*server*[.*screen*]

Specifies the name of the display on which to run *xmbind. host* is the hostname of the physical display, *server* specifies the server number, and *screen* specifies the screen number. Either or both of the *host* and *screen* elements can be omitted. If *host* is omitted, the local display is assumed. If *screen* is omitted, screen 0 is assumed (and the period is unnecessary). The colon and (display) *server* are necessary in all cases.

Environment

The XMBINDDIR environment variable affects the way that *xmbind* searches for vendor-specific default virtual bindings.

See Also

`XmTranslateKey`(1).