# Programs

- mwm
- uil
- xmbind

# mwm (user cmd)

# NAME

**mwm** –The Motif Window Manager

# SYNOPSIS

**mwm** [*options*]

# DESCRIPTION

The **mwm** window manager provides functions that facilitate control (by the user and the programmer) of elements of window state such as placement, size, icon/normal display, and input–focus ownership.

The stand–alone window manager is not an integral part of CDE and does not support communication with other components in the CDE environment, such as the Style Manager and the Session Manager.

## Options

*-display display*

This option specifies the display to use; see **X**(1).

*-xrm resourcestring*

This option specifies a resource string to use.

*-multiscreen*

This option causes **mwm** to manage all screens on the display. Since **mwm** does this by default, this option is of limited use. See the *multiScreen* resource for information on managing a single screen.

*-name name*

This option causes **mwm** to retrieve its resources using the specified name, as in *name*resource*.

*-screens name [name [...]]*

This option specifies the resource names to use for the screens managed by **mwm**. 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. Screen 0 gets the first name, screen 1 the second name, and so on.

## Appearance

The following sections describe the basic default behaviors of windows, icons, the icon box, input focus, and window stacking. The appearance and behavior of the window manager can be altered by changing the configuration of specific resources. Resources are defined under the heading "X DEFAULTS."

## Screens

By default, **mwm** manages only the single screen specified by the *-display* option or the **DISPLAY** environment variable (by default, screen 0). If the *-multiscreen* option is specified or if the *multiScreen* resource is True, **mwm** tries to manage all the screens on the display.

When **mwm** is managing multiple screens, the *-screens* option can be used to give each screen a unique resource name. The names are separated by blanks, for example, *-screens* scr0 scr1. If there are more screens than names, resources for the remaining screens will be retrieved using the first name. By default, the screen number is used for the screen name.

## Windows

Default **mwm** window frames have distinct components with associated functions:

*Title Area*

In addition to displaying the client's title, the title area is used to move the window. To move the window, place the pointer over the title area, press button 1 and drag the window to a new location. By default, a wire frame is moved during the drag to indicate the new location. When the button is released, the window is moved to the new location.

*Title Bar*

The title bar includes the title area, the minimize button, the maximize button, and the window menu button. In shaped windows, such as round windows, the title bar floats above the window.

*Minimize Button*

To turn the window into an icon, click button 1 on the minimize button (the frame box with a *small* square in it).

*Maximize Button*

To make the window fill the screen (or enlarge to the largest size allowed by the configuration files), click button 1 on the maximize button (the frame box with a *large* square in it).

*Window Menu Button*

The window menu button is the frame box with a horizontal bar in it. To pull down the window menu, press button 1. While pressing, drag the pointer on the menu to your selection, then release the button when your selection is highlighted. Pressing button 3 in the title bar or resize border handles also posts the window menu. Alternately, you can click button 1 to pull down the menu and keep it posted; then position the pointer and select. You can also post the window menu by pressing <Shift> <Esc> or <Alt> <Space>. Double−clicking button 1 with the pointer on the window menu button closes the window.

The following table lists the contents of the window menu.

### Default Window Menu

| Selection | Accelerator | Description |
|---|---|---|
| Restore | | Restores the window to its size before minimizing or maximizing. |
| Move | | Allows the window to be moved with keys or mouse. |
| Size | | Allows the window to be resized. |
| Minimize | | Turns the window into an icon. |
| Maximize | | Makes the window fill the screen. |
| Lower | | Moves window to bottom of window stack. |
| Close | Alt+F4 | Causes client to terminate. |

*Resize Border Handles*

To change the size of a window, move the pointer over a resize border handle (the cursor changes), press button 1, and drag the window to a new size. When the button is released, the window is resized. While dragging is being done, a rubber−band outline is displayed to indicate the new window size.

*Matte*

An optional matte decoration can be added between the client area and the window frame (see the *matteWidth* resource). A *matte* is not actually part of the window frame. There is no functionality associated with a matte.

## Icons

Icons are small graphic representations of windows. A window can be minimized (iconified) using the minimize button on the window frame. Icons provide a way to reduce clutter on the screen.

Pressing mouse button 1 when the pointer is over an icon causes the icon's window menu to pop up. Releasing the button (press + release without moving mouse = click) causes the menu to stay posted. The menu contains the following selections:

### Icon Window Menu

| Selection | Accelerator | Description |
|---|---|---|
| Restore | | Opens the associated window. |
| Move | | Allows the icon to be moved with keys. |
| Size | | Inactive (not an option for icons). |
| Minimize | | Inactive (not an option for icons). |
| Maximize | | Opens the associated window and makes it fill the screen. |
| Lower | | Moves icon to bottom of icon stack. |
| Close | Alt+F4 | Removes client from **mwm** management. |

Note that pressing button 3 over an icon also causes the icon's window menu to pop up. To make a menu selection, drag the pointer over the menu and release button 3 when the desired item is highlighted.

Double−clicking button 1 on an icon invokes the **f.restore_and_raise** function and restores the icon's associated window to its previous state. For example, if a maximized window is iconified, double−clicking button 1 restores it to its maximized state. Double−clicking button 1 on the icon box's icon opens the icon box and allows access to the contained icons. (In general, double−clicking a mouse button is a quick way to perform a function.) Pressing <Shift> <Esc> or <Menu> (the pop−up menu key) causes the icon window menu of the currently selected icon to pop up.

### Icon Box

When icons begin to clutter the screen, they can be packed into an icon box. (To use an icon box, **mwm** must be started with the icon box configuration already set.) The icon box is a **mwm** window that holds client icons. It includes one or more scroll bars when there are more window icons than the icon box can show at the same time.

Icons in the icon box can be manipulated with the mouse. The following table summarizes the behavior of this interface. Button actions apply whenever the pointer is on any part of the icon. Note that double−clicking an icon in the icon box invokes the **f.restore_and_raise** function.

| Button | Action | Description |
|---|---|---|
| Button 1 | click | Selects the icon. |
| Button 1 | double−click | Normalizes (opens) the associated window. Raises an already open window to the top of the stack. |
| Button 1 | drag | Moves the icon. |
| Button 3 | press | Causes the menu for that icon to pop up. |
| Button 3 | drag | Highlights items as the pointer moves across the menu. |

Pressing mouse button 3 when the pointer is over an icon causes the menu for that icon to pop up.

### Icon Menu for the Icon Box

| Selection | Accelerator | Description |
|---|---|---|
| Restore | | Opens the associated window (if not already open). |
| Move | | Allows the icon to be moved with keys. |
| Size | | Inactive. |
| Minimize | | Inactive. |
| Maximize | | Opens the associated window (if not already open) and maximizes its size. |
| Lower | | Inactive. |
| Close | Alt+F4 | Removes client from *mwm* management. |

To pull down the window menu for the icon box itself, press button 1 with the pointer over the menu button for the icon box. The window menu of the icon box differs from the window menu of a client window: The "Close" selection is replaced with the "PackIcons Shift+Alt+F7" selection. When selected, PackIcons packs the icons in the box to achieve neat rows with no empty slots.

You can also post the window menu by pressing <Shift>, <Esc> or <Alt> <Space>. Pressing <Menu> (the pop−up menu key) causes the icon window menu of the currently selected icon to pop up.

### Input Focus

The **mwm** window manager supports (by default) a keyboard input focus policy of explicit selection. This means when a window is selected to get keyboard input, it continues to get keyboard input until the window is withdrawn from window management, another window is explicitly selected to get keyboard input, or the window is iconified. Several resources control the input focus. The client window with the keyboard input focus has the active window appearance with a visually distinct window frame.

The following tables summarize the keyboard input focus selection behavior:

| Button | Action | Object | Function Description |
|---|---|---|---|
| Button 1 | press | Window / window frame | Keyboard focus selection. |
| Button 1 | press | Icon | Keyboard focus selection. |

| Key Action | Function Description |
|---|---|
| [Alt][Tab] | Move input focus to next window in window stack (available only in explicit focus mode). |
| [Alt][Shift][Tab] | Move input focus to previous window in window stack (available only in explicit focus mode). |

## Window Stacking

There are two types of window stacks: global window stacks and an application's local family window stack.

The global stacking order of windows may be changed as a result of setting the keyboard input focus, iconifying a window, or performing a window manager window stacking function. When keyboard focus policy is explicit the default value of the *focusAutoRaise* resource is True. This causes a window to be raised to the top of the stack when it receives input focus, for example, by pressing button 1 on the title bar. The key actions defined in the previous table will thus raise the window receiving focus to the top of the stack.

In pointer mode, the default value of *focusAutoRaise* is False, that is, the window stacking order is not changed when a window receives keyboard input focus. The following key actions can be used to cycle through the global window stack.

| Key Action | Function Description |
|---|---|
| [Alt][ESC] | Place top window on bottom of stack. |
| [Alt][Shift][ESC] | Place bottom window on top of stack. |

By default, a window's icon is placed on the bottom of the stack when the window is iconified; however, the default can be changed by the *lowerOnIconify* resource.

Transient windows (secondary windows such a dialog boxes) stay above their parent windows by default; however, an application's local family stacking order may be changed to allow a transient window to be placed below its parent top–level window. The following arguments show the modification of the stacking order for the **f.lower** function.

**f.lower**

Lowers the transient window within the family (staying above the parent) and lowers the family in the global window stack.

**f.lower** [ *within*]

Lowers the transient window within the family (staying above the parent) but does not lower the family in the global window stack.

**f.lower** [*freeFamily* ]

Lowers the window free from its family stack (below the parent), but does not lower the family in the global window stack.

The arguments *within* and *freeFamily* can also be used with **f.raise** and **f.raise_lower**.

## Session Management

The window manager is an X Session Management Protocol aware client. It responds to SaveYourself (and other associated messages) by saving the geometries of its clients to a state file. **mwm** can then be restarted by the XSMP session manager. The default location for the state file is `$HOME/.mwmclientdb`. This location can be overriden with the resource **sessionClientDB**.

## X Resources

The **mwm** command is configured from its resource database. This database is built from the following sources. They are listed in order of precedence, low to high:

`/usr/lib/X11/app-defaults/Mwm`

`$HOME/Mwm`

**RESOURCE_MANAGER** root window property or `$HOME/.Xdefaults`

**XENVIRONMENT** variable or `$HOME/.Xdefaults-host`

*mwm* command line options

The file names `/usr/lib/X11/app-defaults/Mwm` and `$HOME/Mwm` represent customary locations for these files. The actual location of the system−wide class resource file may depend on the **XFILESEARCHPATH** environment variable and the current language environment. The actual location of the user−specific class resource file may depend on the **XUSERFILESEARCHPATH** and **XAPPLRESDIR** environment variables and the current language environment.

Entries in the resource database may refer to other resource files for specific types of resources. These include files that contain bitmaps, fonts, and **mwm** specific resources such as menus and behavior specifications (for example, button and key bindings).

*Mwm* is the resource class name of **mwm** and **mwm** is the default resource name used by **mwm** to look up resources. the *-screens* command line option specifies resource names, such as "mwm_b+w" and "mwm_color".) In the following discussion of resource specification, "Mwm" and "mwm" (and the aliased **mwm** resource names) can be used interchangeably, but "mwm" takes precedence over "Mwm".

The **mwm** command uses the following types of resources:

*Component Appearance Resources:*

> These resources specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, feedback windows (for example, the window reconfiguration feedback window), client window frames, and icons.

*General Appearance and Behavior Resources:*

> These resources specify **mwm** appearance and behavior (for example, window management policies). They are not set separately for different **mwm** user interface components. They apply to all screens and workspaces.

*Screen Specific Appearance and Behavior Resources:*

> These resources specify the appearance and behavior of **mwm** elements that are settable on a per−screen basis.

*Client Specific Resources:*

> These **mwm** resources can be set for a particular client window or class of client windows. They specify client−specific icon and client window frame appearance and behavior.

Resource identifiers can be either a resource name (for example, foreground) or a resource class (for example, Foreground). If the value of a resource is a filename and if the filename is prefixed by "~/", then it is relative to the path contained in the **HOME** environment variable (generally the user's home directory).

## Component Appearance Resources

The syntax for specifying component appearance resources that apply to window manager icons, menus, and client window frames is *Mwm* resource_id*

For example, *Mwm*foreground* is used to specify the foreground color for **mwm** menus, icons, client window frames, and feedback dialogs.

The syntax for specifying component appearance resources that apply to a particular **mwm** component is *Mwm*[menu|icon|client|feedback] *resource_id*

If *menu* is specified, the resource is applied only to **mwm** menus; if *icon* is specified, the resource is applied to icons; and if *client* is specified, the resource is applied to client window frames. For example, *Mwm*icon*foreground* is used to specify the foreground color for **mwm** icons, *Mwm*menu*foreground* specifies the foreground color for **mwm** menus, and *Mwm*client*foreground* is used to specify the foreground color for **mwm** client window frames.

The appearance of the title area of a client window frame (including window management buttons) can be separately configured. The syntax for configuring the title area of a client window frame is *Mwm*client*title* resource_id*

For example, *Mwm*client*title*foreground* specifies the foreground color for the title area. Defaults for title area resources are based on the values of the corresponding client window frame resources.

The appearance of menus can be configured based on the name of the menu. The syntax for specifying menu appearance by name is *Mwm*menu* menu_name*resource_id*

For example, *Mwm*menu*my_menu*foreground* specifies the foreground color for the menu named *my_menu*. The user can also specify

resources for window manager menu components, that is, the gadgets that comprise the menu. These may include for example, a menu title, title separator, one or more buttons, and separators. If a menu contains more than one instance of a class, such as multiple PushButtonGadgets, the name of the first instance is "PushButtonGadget1", the second is "PushButtonGadget2", and so on. The following list identifies the naming convention used for window manager menu components:

- Menu Title LabelGadget - "TitleName"
- Menu Title SeparatorGadget - "TitleSeparator"
- CascadeButtonGadget - "CascadeButtonGadget<n>"
- PushButtonGadget - "PushButtonGadget<n>"
- SeparatorGadget - "SeparatorGadget<n>"

Refer to the man page for each class for a list of resources that can be specified.

The following component appearance resources that apply to all window manager parts can be specified:

### Component Appearance Resources - All Window Manager Parts

| Name | Class | Value Type | Default |
|---|---|---|---|
| background | Background | color | varies[dagger] |
| backgroundPixmap | BackgroundPixmap | string[dagger][dagger] | varies[dagger] |
| bottomShadowColor | Foreground | color | varies[dagger] |
| bottomShadowPixmap | Foreground | string[dagger][dagger] | varies[dagger] |
| fontList | FontList | string[dagger][dagger][dagger] | "fixed" |
| foreground | Foreground | color | varies[dagger] |
| saveUnder | SaveUnder | T/F | F |
| topShadowColor | Background | color | varies[dagger] |
| topShadowPixmap | TopShadowPixmap | string[dagger][dagger] | varies[dagger] |

[dagger]The default is chosen based on the visual type of the screen. [dagger][dagger]Image name. See **XmInstallImage(3)** . [dagger][dagger][dagger]X11 X Logical Font Description

*background* (class *Background*)

This resource specifies the background color. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

*backgroundPixmap* (class *BackgroundPixmap*)

This resource specifies the background Pixmap of the **mwm** decoration when the window is inactive (does not have the keyboard focus). The default value is chosen based on the visual type of the screen.

*bottomShadowColor* (class *Foreground*)

This resource specifies the bottom shadow color. This color is used for the lower and right bevels of the window manager decoration. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

*bottomShadowPixmap* (class *BottomShadowPixmap*)

This resource specifies the bottom shadow Pixmap. This Pixmap is used for the lower and right bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

*fontList* (class *FontList*)

This resource specifies the font used in the window manager decoration. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

*foreground* (class *Foreground*)

This resource specifies the foreground color. The default is chosen based on the visual type of the screen.

*saveUnder* (class *SaveUnder*)

This is used to indicate whether "save unders" are used for **mwm** components. For this to have any effect, save unders must be implemented by the X server. If save unders are implemented, the X server saves the contents of windows obscured by windows that have the save under attribute set. If the *saveUnder* resource is True, **mwm** will set the save under attribute on the window manager frame of any

client that has it set. If *saveUnder* is False, save unders will not be used on any window manager frames. The default value is False.

*topShadowColor* (class *Background*)

This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

**topShadowPixmap (** class *TopShadowPixmap)*

This resource specifies the top shadow Pixmap. This Pixmap is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

The following component appearance resources that apply to frame and icons can be specified:

### Frame and Icon Components

| Name | Class | Value Type | Default |
|---|---|---|---|
| activeBackground | Background | color | varies[dagger] |
| activeBackgroundPixmap | BackgroundPixmap | string[dagger][dagger] | varies[dagger] |
| activeBottomShadowColor | Foreground | color | varies[dagger] |
| activeBottomShadowPixmap | BottomShadowPixmap | string[dagger][dagger] | varies[dagger] |
| activeForeground | Foreground | color | varies[dagger] |
| activeTopShadowColor | Background | color | varies[dagger] |
| activeTopShadowPixmap | TopShadowPixmap | string[dagger][dagger] | varies[dagger] |

[dagger]The default is chosen based on the visual type of the screen. [dagger][dagger]See **XmInstallImage**(3) .

*activeBackground* (class *Background*)

This resource specifies the background color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

*activeBackgroundPixmap* (class *ActiveBackgroundPixmap*)

This resource specifies the background Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

*activeBottomShadowColor* (class *Foreground*)

This resource specifies the bottom shadow color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

*activeBottomShadowPixmap* (class *BottomShadowPixmap*)

This resource specifies the bottom shadow Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

*activeForeground* (class *Foreground*)

This resource specifies the foreground color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

*activeTopShadowColor* (class *Background*)

This resource specifies the top shadow color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

*activeTopShadowPixmap* (class *TopShadowPixmap*)

This resource specifies the top shadow Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

### General Appearance and Behavior Resources

The syntax for specifying general appearance and behavior resources is *Mwm*resource_id*

For example, *Mwm*keyboardFocusPolicy* specifies the window manager policy for setting the keyboard focus to a particular client window.

The following general appearance and behavior resources can be specified:

### General Appearance and Behavior Resources

| Name | Class | Value Type | Default |
|---|---|---|---|
| autoKeyFocus | AutoKeyFocus | T/F | T |
| autoRaiseDelay | AutoRaiseDelay | millisec | 500 |
| bitmap−Directory | Bitmap−Directory | directory | /usr/include−/X11/bitmaps |
| clientAutoPlace | ClientAutoPlace | T/F | T |
| colormapFocus−Policy | ColormapFocus−Policy | string | keyboard |
| configFile | ConfigFile | file | $HOME/mwmrc |
| deiconifyKeyFocus | DeiconifyKeyFocus | T/F | T |
| doubleClick−Time | DoubleClick−Time | millisec. | multi−click time[dagger] |
| enableWarp | enableWarp | T/F | T |
| enforceKeyFocus | EnforceKeyFocus | T/F | T |
| frameStyle | FrameStyle | string | recessed |
| iconAutoPlace | IconAutoPlace | T/F | T |
| iconClick | IconClick | T/F | T |
| interactivePlacement | InteractivePlacement | T/F | T |
| keyboardFocus−Policy | KeyboardFocus−Policy | string | explicit |
| lowerOnIconify | LowerOnIconify | T/F | T |
| moveThreshold | MoveThreshold | pixels | 4 |
| multiScreen | MultiScreen | T/F | T |
| passButtons | PassButtons | T/F | F |
| passSelectButton | PassSelectButton | T/F | T |
| positionIsFrame | PositionIsFrame | T/F | T |
| positionOnScreen | PositionOnScreen | T/F | T |
| quitTimeout | QuitTimeout | millisec. | 1000 |
| raiseKeyFocus | RaiseKeyFocus | T/F | F |
| refreshByClearing | RefreshByClearing | T/F | T |
| rootButtonClick | RootButtonClick | T/F | F |
| screens | Screens | string | varies |
| sessionClientDB | SessionClientDB | string | `$HOME/.mwmclientdb` |
| showFeedback | ShowFeedback | string | all |
| startupKeyFocus | StartupKeyFocus | T/F | T |
| wMenuButtonClick | WMenuButtonClick | T/F | T |
| wMenuButtonClick2 | WMenuButtonClick2 | T/F | T |

[dagger]The resource doubleClickTime is included for backward compatibility. Use of the Xt resource multiClickTime is preferred.

*autoKeyFocus* (class *AutoKeyFocus*)

This resource is available only when the keyboard input focus policy is explicit. If *autoKeyFocus* is given a value of True, then when a window with the keyboard input focus is withdrawn from window management or is iconified, the focus is set to the previous window that had the focus. If the value given is False, there is no automatic setting of the keyboard input focus. It is recommended that both *autoKeyFocus* and *startupKeyFocus* be True to work with tear off menus. The default value is True.

*autoRaiseDelay* (class *AutoRaiseDelay*)

This resource is available only when the *focusAutoRaise* resource is True and the keyboard focus policy is pointer. The *autoRaiseDelay* resource specifies the amount of time (in milliseconds) that **mwm** will wait before raising a window after it gets the keyboard focus. The default value of this resource is 500 (ms).

*bitmapDirectory* (class *BitmapDirectory*)

This resource identifies a directory to be searched for bitmaps referenced by **mwm** resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is BR /usr/include/X11/bitmaps. The directory `/usr/include/X11/bitmaps` represents the customary locations for this directory. The actual location of this directory may vary on some systems. If the bitmap is not found in the specified directory, **XBMLANGPATH** is searched.

*clientAutoPlace* (class *ClientAutoPlace*)

This resource determines the position of a window when the window has not been given a program−specified or user−specified position. With a value of True, windows are positioned with the top left corners of the frames offset horizontally and vertically. A value of False causes the currently configured position of the window to be used. In either case, **mwm** will attempt to place the windows totally on−screen. The default value is True.

*colormapFocusPolicy* (class *ColormapFocusPolicy*)

This resource indicates the colormap focus policy that is to be used. If the resource value is explicit, a colormap selection action is done on a client window to set the colormap focus to that window. If the value is pointer, the client window containing the pointer has the colormap focus. If the value is keyboard, the client window that has the keyboard input focus has the colormap focus. The default value for this resource is keyboard.

*configFile* (class *ConfigFile*)

The resource value is the pathname for a **mwm** resource description file. If the pathname begins with "~/", **mwm** considers it to be relative to the user's home directory (as specified by the **HOME** environment variable). If the **LANG** environment variable is set, **mwm** looks for *$HOME/$LANG/ configFile*. If that file does not exist or if **LANG** is not set, **mwm** looks for `$HOME/`*configFile*. If the *configFile* pathname does not begin with "~/" or "/", **mwm** considers it to be relative to the current working directory. If the *configFile* resource is not specified or if that file does not exist, **mwm** uses several default paths to find a configuration file. The order of the search is shown below: `$HOME/.dt/$LANG/mwmrc $HOME/.dt/mwmrc /etc/dt/config/$LANG/sys.mwmrc` [dagger] `/etc/dt/config/sys.mwmrc` [dagger] `/usr/dt/config/$LANG/sys.mwmrc` [dagger] `/usr/dt/config/sys.mwmrc` [dagger] Paths marked with '[dagger]' are implementation dependent.

*deiconifyKeyFocus* (class *DeiconifyKeyFocus*)

This resource applies only when the keyboard input focus policy is explicit. If a value of True is used, a window receives the keyboard input focus when it is normalized (deiconified). True is the default value.

*doubleClickTime* (class *DoubleClickTime*)

This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double−click. The use of this resource is deprecated. Use the Xt resource *multiClickTime* instead. The value of *doubleClickTime* dynamically defaults to the value of *multiClickTime*.

*enableWarp* (class *EnableWarp*)

The default value of this resource, True, causes **mwm** to warp the pointer to the center of the selected window during keyboard−controlled resize and move operations. Setting the value to False causes **mwm** to leave the pointer at its original place on the screen, unless the user explicitly moves it with the cursor keys or pointing device.

*enforceKeyFocus* (class *EnforceKeyFocus*)

If this resource is given a value of True, the keyboard 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 scroll bar that can be operated without setting the focus to that client.) If the resource is False, the keyboard input focus is not explicitly set to globally active windows. The default value is True.

*frameStyle* (class *frameStyle*)

If this resource is given a value of "slab", the the window manager frame is drawn such that the client area appears to be at the same height as the top of the window frame. If the resource is set to "recessed", the window frame is drawn such that the client area appears lower than the top of the window frame. The default value is "recessed".

*iconAutoPlace* (class *IconAutoPlace*)

This resource indicates 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. The value True indicates that icons are arranged in a particular area of the screen, determined by the *iconPlacement* resource. The value False indicates that an icon is placed at the location of the window when it is iconified. The default is True.

*iconClick* (class *IconClick*)

When this resource is given the value of True, the system menu is posted and left posted when an icon is clicked. The default value is

True.

*interactivePlacement* (class *InteractivePlacement*)

This resource controls the initial placement of new windows on the screen. If the value is True, the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper−left hand corner of the window. If the value is False, windows are placed according to the initial window configuration attributes. The default value of this resource is False.

*keyboardFocusPolicy* (class *KeyboardFocusPolicy*)

If set to pointer, the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that **mwm** adds). If set to explicit, the policy is to have the keyboard focus set to a client window when the user presses button 1 with the pointer on the client window or any part of the associated **mwm** decoration. The default value for this resource is explicit.

*lowerOnIconify* (class *LowerOnIconify*)

If this resource is given the default value of True, a window's icon appears on the bottom of the window stack when the window is minimized (iconified). A value of False places the icon in the stacking order at the same place as its associated window. The default value of this resource is True.

*moveThreshold* (class *MoveThreshold*)

This resource is used to control the sensitivity of dragging operations that move windows and icons. The value of this resource is the number of pixels that the locator is moved with a button down before the move operation is initiated. This is used to prevent window/icon movement when you click or double−click and there is unintentional pointer movement with the button down. The default value of this resource is 4 (pixels).

*multiScreen* (class *MultiScreen*)

This resource, if True, causes **mwm** to manage all the screens on the display. If False, **mwm** manages only a single screen. The default value is True.

*passButtons* (class *PassButtons*)

This resource indicates whether or not button press events are passed to clients after they are used to do a window manager function in the client context. If the resource value is False, the button press is not passed to the client. If the value is True, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is False.

*passSelectButton* (class *PassSelectButton*)

This resource indicates whether or not to pass the select button press events to clients after they are used to do a window manager function in the client context. If the resource value is False, then the button press will not be passed to the client. If the value is True, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is True.

*positionIsFrame* (class *PositionIsFrame*)

This resource indicates how client window position information (from the *WM_NORMAL_HINTS* property and from configuration requests) is to be interpreted. If the resource value is True, the information is interpreted as the position of the **mwm** client window frame. If the value is False, it is interpreted as being the position of the client area of the window. The default value of this resource is True.

*positionOnScreen* (class *PositionOnScreen*)

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

*quitTimeout* (class *QuitTimeout*)

This resource 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 value of this resource is 1000 (ms). (Refer to the **f.kill** function description for additional information.)

*raiseKeyFocus* (class *RaiseKeyFocus*)

This resource is available only when the keyboard input focus policy is explicit. When set to True, this resource specifies that a window raised by means of the **f.normalize_and_raise** function also receives the input focus. The default value of this resource is False.

*refreshByClearing* (class *RefreshByClearing*)

This resource determines the mechanism used to refresh a window (or the screen) when the **f.refresh_win** (**f.refresh**) function is executed. When set to True, an XClearArea is performed over the window for **f.refresh_win**. When set to False, a covering window is created and destroyed over the top of the window to be refreshed. If the function is **f.refresh** and this resource is set to True, then an XClearArea is performed over every window on the screen. If the resource is set to False, then one large window covering the entire screen is created and destroyed. The default value of this resource is True.

*rootButtonClick* (class *RootButtonClick*)

The *rootButtonClick* resource controls whether the a click on the root window will post the root menu in a "sticky" mode. If this resource is set to True, a button click on the root window will post the menu bound to the button down event for that button in a "sticky" fashion. If this resource is set to False, then the same button click would only cause the menu to flash as it would be unposted once the button up event is seen. The criterion used to determine if it is a button click is if the pointer doesn't move between the button down and button up events. The default value for this resource is True.

*screens* (class *Screens*)

This resource specifies the resource names to use for the screens managed by **mwm**. 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. Screen 0 gets the first name, screen 1 the second name, and so on. The default screen names are 0, 1, and so on.

*sessionClientDB* (class *SessionClientDB*)

This resource identifies a file name to use as a root when saving state at the request of an XSMP session manager. When the session is saved, the window manager will then reuse the file name by automatically incrementing a suffix.

*showFeedback* (class *ShowFeedback*)

This resource controls whether or not feedback windows or confirmation dialogs are displayed. A feedback window shows a client window's initial placement and shows position and size during move and resize operations. Confirmation dialogs can be displayed for certain operations. The value for this resource is a list of names of the feedback options to be enabled or disabled; the names must be separated by a space. If an option is preceded by a minus sign, that option is excluded from the list. The *sign* of the first item in the list determines the initial set of options. If the sign of the first option is minus, **mwm** assumes all options are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), **mwm** starts with no options and builds up a list from the resource.

The names of the feedback options are shown below:

| Name | Description |
|---|---|
| all | Show all feedback (Default value). |
| behavior | Confirm behavior switch. |
| kill | Confirm on receipt of KILL signal. |
| move | Show position during move. |
| none | Show no feedback. |
| placement | Show position and size during initial placement. |
| quit | Confirm quitting **mwm**. |
| resize | Show size during resize. |
| restart | Confirm *mwm restart*. |

The following command line illustrates the syntax for showFeedback:

```
Mwm*showFeedback: placement resize behavior restart
```

This resource specification provides feedback for initial client placement and resize, and enables the dialog boxes to confirm the restart and set behavior functions. It disables feedback for the move function. The default value for this resource is all.

*startupKeyFocus* (class *StartupKeyFocus*)

This resource is available only when the keyboard input focus policy is explicit. When given the default value of True, a window gets the keyboard input focus when the window is mapped (that is, initially managed by the window manager). It is recommended that both *autoKeyFocus* and *startupKeyFocus* be True to work with tear off menus. The default value is True.

*wMenuButtonClick* (class *WMenuButtonClick*)

This resource indicates whether a click of the mouse when the pointer is over the window menu button posts and leaves posted the window menu. If the value given this resource is True, the menu remains posted. True is the default value for this resource.

*wMenuButtonClick2* (class *WMenuButtonClick2*)

When this resource is given the default value of True, a double−click action on the window menu button does an *f.kill function.*

## Screen Specific Appearance and Behavior Resources

The syntax for specifying screen specific resources is *Mwm* screen_name*resource_id* For example, *Mwm*1*keyBindings* specifies the key bindings to use for screen "1".

## Screen Specific Resources

| Name | Class | Value Type | Default |
|---|---|---|---|
| buttonBindings | ButtonBindings | string | DefaultButtonBindings |
| cleanText | CleanText | T/F | T |
| fadeNormalIcon | FadeNormalIcon | T/F | F |
| feedbackGeometry | FeedbackGeometry | string | center on screen |
| frameBorderWidth | FrameBorderWidth | pixels | varies |
| iconBoxGeometry | IconBoxGeometry | string | 6x1+0−0 |
| iconBoxName | IconBoxName | string | iconbox |
| iconBoxSBDisplayPolicy | IconBoxSBDisplayPolicy | string | all |
| iconBoxTitle | IconBoxTitle | XmString | Icons |
| iconDecoration | IconDecoration | string | varies |
| iconImageMaximum | IconImageMaximum | wxh | 48x48 |
| iconImageMinimum | IconImageMinimum | wxh | 16x16 |
| iconPlacement | IconPlacement | string | left bottom |
| iconPlacementMargin | IconPlacementMargin | pixels | varies |
| keyBindings | KeyBindings | string | DefaultKeyBindings |
| limitResize | LimitResize | T/F | T |
| maximumMaximumSize | MaximumMaximumSize | wxh (pixels) | 2X screen w&h |
| moveOpaque | MoveOpaque | T/F | F |
| resizeBorderWidth | ResizeBorderWidth | pixels | varies |
| resizeCursors | ResizeCursors | T/F | T |
| transientDecoration | TransientDecoration | string | menu title |
| transientFunctions | TransientFunctions | string | -minimize-maximize |
| useIconBox | UseIconBox | T/F | F |

*buttonBindings* (class *ButtonBindings*)

This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the **mwm** resource description file. These button bindings are *merged* with the built−in default bindings. The default value for this resource is "DefaultButtonBindings".

*cleanText* (class *CleanText*)

This resource controls the display of window manager text in the client title and feedback windows. If the default value of True is used, the text is drawn with a clear (no stipple) background. This makes text easier to read on monochrome systems where a backgroundPixmap is specified. Only the stippling in the area immediately around the text is cleared. If False, the text is drawn directly on top of the existing background.

*fadeNormalIcon* (class *FadeNormalIcon*)

If this resource is given a value of True, an icon is grayed out whenever it has been normalized (its window has been opened). The default value is False.

*feedbackGeometry* (class *FeedbackGeometry*)

This resource sets the position of the move and resize feedback window. If this resource is not specified, the default is to place the feedback window at the center of the screen. The value of the resource is a standard window geometry string with the following syntax: [=]{ +−}*xoffset*{+−} *yoffset*]

*frameBorderWidth* (class *FrameBorderWidth*)

This resource specifies the width (in pixels) of a client window frame border without resize handles. The border width includes the 3–D shadows. The default value is based on the size and resolution of the screen.

*iconBoxGeometry* (class *IconBoxGeometry*)

This resource indicates the initial position and size of the icon box. The value of the resource is a standard window geometry string with the following syntax: [=][*width xheight*][{+−}*xoffset* {+−}*yoffset*] If the offsets are not provided, the iconPlacement policy is used to determine the initial placement. The units for width and height are columns and rows. The actual screen size of the icon box window depends on the iconImageMaximum (size) and *iconDecoration* resources. The default value for size is (6 * iconWidth + padding) wide by (1 * iconHeight + padding) high. The default value of the location is +0 –0.

*iconBoxName* (class *IconBoxName*)

This resource specifies the name that is used to look up icon box resources. The default name is iconbox.

*iconBoxSBDisplayPolicy* (class *IconBoxSBDisplayPolicy*)

This resource specifies the scroll bar display policy of the window manager in the icon box. The resource has three possible values: all, vertical, and horizontal. The default value, "all", causes both vertical and horizontal scroll bars always to appear. The value "vertical" causes a single vertical scroll bar to appear in the icon box and sets the orientation of the icon box to horizontal (regardless of the iconBoxGeometry specification). The value "horizontal" causes a single horizontal scroll bar to appear in the icon box and sets the orientation of the icon box to vertical (regardless of the iconBoxGeometry specification).

*iconBoxTitle* (class *IconBoxTitle*)

This resource specifies the name that is used in the title area of the icon box frame. The default value is Icons.

*iconDecoration* (class *IconDecoration*)

This resource specifies the general icon decoration. The resource value is label (only the label part is displayed) or image (only the image part is displayed) or label image (both the label and image parts are displayed). A value of activelabel can also be specified to get a label (not truncated to the width of the icon) when the icon is selected. The default icon decoration for icon box icons is that each icon has a label part and an image part (label image). The default icon decoration for stand alone icons is that each icon has an active label part, a label part, and an image part (activelabel label image).

*iconImageMaximum* (class *IconImageMaximum*)

This resource specifies the maximum size of the icon image. The resource value is *widthx height* (for example, 64x64). The maximum supported size is 128x128. The default value of this resource is 50x50.

*iconImageMinimum* (class *IconImageMinimum*)

This resource specifies the minimum size of the icon image. The resource value is *widthx height* (for example, 32x50). The minimum supported size is 16x16. The default value of this resource is 16x16.

*iconPlacement* (class *IconPlacement*)

This resource specifies the icon placement scheme to be used. The resource value has the following syntax:

*primary_layout secondary_layout [tight]*

The layout values are one of the following:

| Value | Description |
|---|---|
| top | Lay the icons out top to bottom. |
| bottom | Lay the icons out bottom to top. |
| left | Lay the icons out left to right. |
| right | Lay the icons out right to left. |

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

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

The default placement is left bottom (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen). A *tight* value places icons with zero spacing in between icons. This value is useful for aesthetic reasons, as well as X−terminals with small screens.

*iconPlacementMargin* (class *IconPlacementMargin*)

This resource sets the distance between the edge of the screen and the icons that are placed along the edge of the screen. The value should be greater than or equal to 0. A default value (see below) is used if the value specified is invalid. The default value for this resource is equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column).

*keyBindings* (class *KeyBindings*)

This resource identifies the set of key bindings for window management functions. If specified, these key bindings *replace* the built−in default bindings. The named set of key bindings is specified in **mwm** resource description file. The default value for this resource is "DefaultKeyBindings".

*limitResize* (class *LimitResize*)

If this resource is True, the user is not allowed to resize a window to greater than the maximum size. The default value for this resource is True.

*maximumMaximumSize* (class *MaximumMaximumSize*)

This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is *widthxheight* (for example, 1024x1024) where the width and height are in pixels. The default value of this resource is twice the screen width and height.

*moveOpaque* (class *MoveOpaque*)

This resource controls whether the actual window is moved or a rectangular outline of the window is moved. A default value of False displays a rectangular outline on moves.

*resizeBorderWidth* (class *ResizeBorderWidth*)

This resource specifies the width (in pixels) of a client window frame border with resize handles. The specified border width includes the 3 −D shadows. The default value is based on the size and resolution of the screen.

*resizeCursors* (class *ResizeCursors*)

This is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If True, the cursors are shown, otherwise the window manager cursor is shown. The default value is True.

*transientDecoration* (class *TransientDecoration*)

This controls the amount of decoration that **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 value for this resource is menu title (that is, transient windows have frame borders and a titlebar with a window menu button).

An application can also specify which decorations **mwm** should apply to its windows. If it does so, **mwm** applies only those decorations indicated by both the application and the *transientDecoration* resource. Otherwise, **mwm** applies the decorations indicated by the *transientDecoration* resource. For more information see the description of **XmNmwmDecorations** on the **VendorShell**(3) reference page.

*transientFunctions* (class *TransientFunctions*)

This resource is used to indicate 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 value for this resource is −minimize −maximize.

An application can also specify which functions **mwm** should apply to its windows. If it does so, **mwm** applies only those functions indicated by both the application and the *transientFunctions* resource. Otherwise, **mwm** applies the functions indicated by the *transientFunctions* resource. For more information see the description of **XmNmwmFunctions** on the **VendorShell**(3) reference page.

*useIconBox* (class *UseIconBox*)

If this resource is given a value of True, icons are placed in an icon box. When an icon box is not used, the icons are placed on the root window (default value).

## Client Specific Resources

The syntax for specifying client specific resources is

*Mwm*client_name_or_class *resource_id*

For example, *Mwm*mterm*windowMenu* is used to specify the window menu to be used with mterm clients. The syntax for specifying client specific resources for all classes of clients is

*Mwm*resource_id*

Specific client specifications take precedence over the specifications for all clients. For example, *Mwm*windowMenu* is used to specify the window menu to be used for all classes of clients that don't have a window menu specified.

The syntax for specifying resource values for windows that have an unknown name and class (that is, windows that do not have a **WM_CLASS** property associated with them) is

*Mwm*defaults*resource_id*

For example, *Mwm*defaults*iconImage* is used to specify the icon image to be used for windows that have an unknown name and class.

The following client specific resources can be specified:

## Client Specific Resources

| Name | Class | Value Type | Default |
|---|---|---|---|
| clientDecoration | ClientDecoration | string | all. |
| clientFunctions | ClientFunctions | string | all. |
| focusAutoRaise | FocusAutoRaise | T/F | varies |
| iconImage | IconImage | pathname | (image) |
| iconImage–Background | Background | color | icon background |
| iconImageBottom–ShadowColor | Foreground | color | icon bottom shadow |
| iconImageBottom–ShadowPixmap | BottomShadow–Pixmap | color | icon bottom shadow pixmap |
| iconImageForeground | Foreground | color | varies |
| iconImageTopShadowColor | Background | color | icon top shadow color |
| iconImageTop–ShadowPixmap | TopShadowPixmap | color | icon top shadow pixmap |
| matteBackground | Background | color | background |
| matteBottom–ShadowColor | Foreground | color | bottom shadow color |
| matteBottom–ShadowPixmap | BottomShadow–Pixmap | color | bottom shadow pixmap |
| matteForeground | Foreground | color | foreground |
| matteTopShadowColor | Background | color | top shadow color |
| matteTopShadowPixmap | TopShadowPixmap | color | top shadow pixmap |
| matteWidth | MatteWidth | pixels | 0 |
| maximumClientSize | MaximumClientSize | wxh vertical horizontal | fill the screen |
| useClientIcon | UseClientIcon | T/F | T |
| usePPosition | UsePPosition | string | nonzero |
| windowMenu | WindowMenu | string | DefaultWindowMenu |

*clientDecoration* (class *ClientDecoration*)

This resource controls the amount of window frame decoration. The resource is specified as a list of decorations to specify their inclusion in the frame. If a decoration is preceded by a minus sign, that decoration is excluded from the frame. The *sign* of the first item in the list

determines the initial amount of decoration. If the sign of the first decoration is minus, **mwm** assumes all decorations are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), then **mwm** starts with no decoration and builds up a list from the resource.

An application can also specify which decorations **mwm** should apply to its windows. If it does so, **mwm** applies only those decorations indicated by both the application and the *clientDecoration* resource. Otherwise, **mwm** applies the decorations indicated by the *clientDecoration* resource. For more information see the description of **XmNmwmDecorations** on the **VendorShell**(3) reference page.

| Name | Description |
|---|---|
| all | Include all decorations (default value). |
| border | Window border. |
| maximize | Maximize button (includes title bar). |
| minimize | Minimize button (includes title bar). |
| none | No decorations. |
| resizeh | Border resize handles (includes border). |
| menu | Window menu button (includes title bar). |
| title | Title bar (includes border). |

Examples: *Mwm*XClock.clientDecoration: −resizeh −maximize* This removes the resize handles and maximize button from XClock windows. *Mwm*XClock.clientDecoration: menu minimize border* This does the same thing as above. Note that either *menu* or *minimize* implies *title*.

*clientFunctions* (class *ClientFunctions*)

This resource is used to indicate which **mwm** functions are applicable (or not applicable) to the client window. The value for the resource is a list of functions. If the first function in the list has a minus sign in front of it, then **mwm** starts with all functions and subtracts from that set. If the first function in the list has a plus sign in front of it, then **mwm** starts with no functions and builds up a list. Each function in the list must be preceded by the appropriate plus or minus sign and separated from the next function by a space.

An application can also specify which functions **mwm** should apply to its windows. If it does so, **mwm** applies only those functions indicated by both the application and the *clientFunctions* resource. Otherwise, **mwm** applies the functions indicated by the *clientFunctions* resource. For more information see the description of **XmNmwmFunctions** on the **VendorShell**(3) reference page.

The following table lists the functions available for this resource:

| Name | Description |
|---|---|
| all | Include all functions (default value). |
| none | No functions. |
| resize | f.resize[dagger]. |
| move | f.move[dagger]. |
| minimize | f.minimize[dagger]. |
| maximize | f.maximize[dagger]. |
| close | f.kill[dagger]. |

[dagger]See **mwmrc**(4) .

*focusAutoRaise* (class *FocusAutoRaise*)

When the value of this resource is True, clients are raised when they get the keyboard input focus. If the value is False, the stacking of windows on the display is not changed when a window gets the keyboard input focus. The default value is True when the keyboardFocusPolicy is explicit and False when the keyboardFocusPolicy is pointer.

*iconImage* (class *IconImage*)

This resource can be used to specify an icon image for a client (for example, "Mwm*myclock*iconImage"). The resource value is a pathname for a pixmap or bitmap file. The value of the (client specific) *useClientIcon* resource is used to determine whether or not user supplied icon images are used instead of client supplied icon images. The default value is to display a built−in window manager icon image.

*iconImageBackground* (class *Background*)

This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (that is, specified by "Mwm*background or Mwm*icon*background).

*iconImageBottomShadowColor* (class *Foreground*)

This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this

resource is the icon bottom shadow color (that is, specified by Mwm*icon*bottomShadowColor).

*iconImageBottomShadowPixmap* (class *BottomShadowPixmap*)

This resource specifies the bottom shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow Pixmap (that is, specified by Mwm*icon*bottomShadowPixmap).

*iconImageForeground* (class *Foreground*)

This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this resource varies depending on the icon background.

*iconImageTopShadowColor* (class *Background*)

This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color (that is, specified by Mwm*icon*topShadowColor).

*iconImageTopShadowPixmap* (class *TopShadowPixmap*)

This resource specifies the top shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow pixmap (that is, specified by Mwm*icon*topShadowPixmap).

*matteBackground* (class *Background*)

This resource specifies the background color of the matte, when *matteWidth* is positive. The default value of this resource is the client background color (that is, specified by "Mwm*background or Mwm*client*background).

*matteBottomShadowColor* (class *Foreground*)

This resource specifies the bottom shadow color of the matte, when *matteWidth* is positive. The default value of this resource is the client bottom shadow color (that is, specified by Mwm*bottomShadowColor or Mwm*client*bottomShadowColor).

*matteBottomShadowPixmap* (class *BottomShadowPixmap*)

This resource specifies the bottom shadow Pixmap of the matte, when *matteWidth* is positive. The default value of this resource is the client bottom shadow pixmap (that is, specified by Mwm*bottomShadowPixmap or Mwm*client*bottomShadowPixmap).

*matteForeground* (class *Foreground*)

This resource specifies the foreground color of the matte, when *matteWidth* is positive. The default value of this resource is the client foreground color (that is, specified by Mwm*foreground or Mwm*client*foreground).

*matteTopShadowColor* (class *Background*)

This resource specifies the top shadow color of the matte, when *matteWidth* is positive. The default value of this resource is the client top shadow color (that is, specified by Mwm*topShadowColor or Mwm*client*topShadowColor).

*matteTopShadowPixmap* (class *TopShadowPixmap*)

This resource specifies the top shadow pixmap of the matte, when *matteWidth* is positive. The default value of this resource is the client top shadow pixmap (that is, specified by "Mwm*topShadowPixmap or Mwm*client*topShadowPixmap).

*matteWidth* (class *MatteWidth*)

This resource specifies the width of the optional matte. The default value is 0, which effectively disables the matte.

*maximumClientSize* (class *MaximumClientSize*)

This resource is either a size specification or a direction that indicates how a client window is to be maximized. The resource value can be specified as a size specification *widthxheight*. The width and height are interpreted in the units that the client uses (for example, for terminal emulators this is generally characters). Alternately, "vertical" or "horizontal" can be specified to indicate the direction in which the client maximizes.

If this resource is not specified, the maximum size from the *WM_NORMAL_HINTS* property is used if set. Otherwise the default value is the size where the client window with window management borders fills the screen. When the maximum client size is not determined by

the *maximumClientSize* resource, the *maximumMaximumSize* resource value is used as a constraint on the maximum size.

*useClientIcon* (class *UseClientIcon*)

If the value given for this resource is True, a client−supplied icon image takes precedence over a user−supplied icon image. The default value is True, giving the client−supplied icon image higher precedence than the user−supplied icon image.

*usePPosition* (class *UsePPosition*)

This resource specifies whether Mwm honors program specified position **PPosition** specified in the *WM_NORMAL_HINTS* property in the absence of an user specified position. Setting this resource to on, causes **mwm** to always honor program specified position. Setting this resource to off, causes **mwm** to always ignore program specified position. Setting this resource to the default value of nonzero cause **mwm** to honor program specified position other than (0,0).

*windowMenu* (class *WindowMenu*)

This resource indicates the name of the menu pane that is posted when the window menu is popped up (usually by pressing button 1 on the window menu button on the client window frame). Menu panes are specified in the **mwm** resource description file. Window menus can be customized on a client class basis by creating custom menus in your *mwmrc* file (see **mwmrc**(4) and specifying resources to activate the custom menus. The resources have the form *Mwm* client_name_or_class*windowMenu*. The default value of this resource is DefaultWindowMenu.

## Resource Description File

The **mwm** resource description file is a supplementary resource file that contains resource descriptions that are referred to by entries in the resource manager property (see **xrdb**(1) and the defaults files (`.Xdefaults`, *app−defaults/Mwm* ). It contains descriptions of resources that are to be used by **mwm**, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular **mwm** resource description file can be selected using the *configFile* resource.

The following types of resources can be described in the **mwm** resource description file:

*Buttons*

Window manager functions can be bound (associated) with button events.

*Keys*

Window manager functions can be bound (associated) with key press events.

*Menus*

Menu panes can be used for the window menu and other menus posted with key bindings and button bindings.

The **mwm** resource description file is described in **mwmrc**(4) .

## Environment

The **mwm** window manager uses the environment variable **HOME** specifying the user's home directory.

The **mwm** window manager uses the environment variable **LANG** specifying the user's choice of language for the **mwm** message catalog and the **mwm** resource description file.

The **mwm** window uses the environment variable **XFILESEARCHPATH**, **XUSERFILESEARCHPATH**, **XAPPLRESDIR**, **XENVIRONMENT**, **LANG**, and **HOME** in determining search paths for resource defaults files. The **mwm** window manager may also us **XBMLANGPATH** to search for bitmap files.

The **mwm** window manager reads the `$HOME/.motifbind` file if it exists to install a virtual key bindings property on the root window. For more information on the content of the `.motifbind` file, see

The **mwm** window manager uses the environment variable **MWMSHELL** (or **SHELL**, *if* **MWMSHELL** is not set), specifying the shell to use when executing commands via the **f.exec** function.

### 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

### RELATED INFORMATION

**VendorShell**(3) , **VirtualBindings**(3) , **X**(1), **XmInstallImage**(3) , **xrdb**(1).

# uil (user cmd)

# NAME

**uil** –The user interface language compiler

# SYNOPSIS

**uil** [ [*options*] ] [*file*]

# DESCRIPTION

The **uil** command invokes the UIL compiler. The User Interface Language (UIL) is a specification language for describing the initial state of a user interface for a Motif application. The specification describes the objects (menus, dialog boxes, labels, push buttons, and so on) used in the interface and specifies the routines to be called when the interface changes state as a result of user interaction.

*file*

Specifies the file to be compiled through the UIL compiler.

*options*

Specifies one or more of the following options:

*-Ipathname*

This option causes the compiler to look for include files in the directory specified if the include files have not been found in the paths that already were searched. Specify this option followed by a pathname, with no intervening spaces.

*-m*

Machine code is listed. This directs the compiler to place in the listing file a description of the records that it added to the User Interface Database (UID). This helps you isolate errors. The default is no machine code.

*-o file*

Directs the compiler to produce a UID. By default, UIL creates a UID with the name *a.uid*. The file specifies the filename for the UID. No UID is produced if the compiler issues any diagnostics categorized as error or severe. UIDs are portable only across same −size machine architectures.

*-s*

Directs the compiler to set the locale before compiling any files. The locale is set in an implementation−dependent manner. On ANSI C−based systems, the locale is usually set by calling **setlocale(LC_ALL, "")**. If this option is not specified, the compiler does not set the locale.

*-v file*

Directs the compiler to generate a listing. The file specifies the filename for the listing. If the *-v* option is not present, no listing is generated by the compiler. The default is no listing.

*-w*

Specifies that the compiler suppress all warning and informational messages. If this option is not present, all messages are generated, regardless of the severity.

*-wmd file*

Specifies a binary widget meta−language description file to be used in place of the default WML description.

## RELATED INFORMATION

**X**(1) and **Uil(3)** .

## xmbind (user cmd)

# NAME

**xmbind** - Configures virtual key bindings

# SYNOPSIS

**xmbind** [ [*options*] ] [ [*file*] ]

# DESCRIPTION

**xmbind** is an X Window System client that configures the virtual key bindings for Motif applications. This action is performed by **mwm** at its startup, so the **xmbind** client is only needed when **mwm** is not in use, or when you want to change bindings without restarting **mwm**. If a file is specified, its contents are used as the virtual key bindings. If a file is not specified, the file *.motifbind* in the user's home directory is used. If this file is not found, **xmbind** loads the default virtual key bindings, as described in **VirtualBindings(3)** .

### Options

*-display*

This option specifies the display to use; see X(1).

# RELATED

**VirtualBindings**(3) and **X**(1).