NAME

mwmrc — the Motif Window Manager Resource Description File

DESCRIPTION

The **mwmrc** file is a supplementary resource file that controls much of the behavior of the Motif window manager **mwm**. It contains descriptions of resources that cannot easily be written using standard X Window System, Version 11 resource syntax. The resource description file contains entries that are referred to by X resources in defaults files (for example, **/usr/X11R6/lib/X11/app-defaults/Mwm**) or in the **RESOURCE_MANAGER** property on the root window. For example, the resource description file enables you to specify different types of window menus; however, an X resource is used to specify which of these window menus **mwm** should use for a particular window.

Location

The window manager searches for one of the following resource description files, where *$LANG* is the value of the language environment on a per-user basis:

```
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
/usr/X11R6/lib/X11/$LANG/system.mwmrc
/usr/X11R6/lib/X11/system.mwmrc
```

The first file found is the first used. If no file is found, a set of built-in specifications is used. A particular resource description file can be selected using the *configFile* resource. The following shows how a different resource description file can be specified from the command line:

```
/usr/X11R6/bin/X11/mwm -xrm "mwm*configFile: mymwmrc"
```

Resource Types

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.

MWM RESOURCE DESCRIPTION FILE SYNTAX

The **mwm** resource description file is a standard text file that contains items of information separated by blanks, tabs, and new lines characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (for example, the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes (" "). Single characters can be quoted by preceding them by the back-slash character (\). If a line ends with a back-slash, the next line is considered a continuation of that line. All text from an unquoted **#** to the end of the line is regarded as a comment and is not interpreted as part of a resource description. If **!** is the first character in a line, the line is regarded as a comment.

Window Manager Functions

Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for button and key binding sets, and menu panes. The function specification has the following syntax:

function = function_name [function_args]
function_name = window manager function
function_args = {quoted_item | unquoted_item}

The following functions are supported. If a function is specified that isn't one of the supported functions then it is interpreted by **mwm** as **f.nop**.

f.beep This function causes a beep.

f.circle_down [*icon* | *window*]

This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it is no longer obscuring any other window or icon). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an *icon* function argument is specified, then the function applies only to icons. If a *window* function argument is specified then the function applies only to windows.

f.circle_up [*icon* | *window*]

This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (that is, transient windows) are restacked with their associated primary window. If an *icon* function argument is specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.exec command (or *!* **command**)

This function causes *command* to be executed (using the value of the *$MWMSHELL* or *$SHELL* environment variable if set; otherwise, **/bin/sh**). The *!* notation can be used in place of the **f.exec** function name.

f.focus_color

This function sets the colormap focus to a client window. If this function is done in a root context, then the default colormap (setup by the X Window System for the screen where **mwm** is running) is installed and there is no specific client window colormap focus. This function is treated as **f.nop** if *colormapFocusPolicy* is not explicit.

f.focus_key

This function sets the keyboard input focus to a client window or icon. This function is treated as **f.nop** if *keyboardFocusPolicy* is not explicit or the function is executed in a root context.

f.kill This function is used to close application windows. The actual processing that occurs depends on the protocols that the application observes. The application lists the protocols it observes in the **WM_PROTOCOLS** property on its top level window. If the application observes the **WM_DELETE_WINDOW** protocol, it is sent a message that requests the window be deleted. If the application observes both **WM_DELETE_WINDOW** and **WM_SAVE_YOURSELF**, it is sent one message requesting the window be deleted and another message advising it to save its state. If the application observes only the **WM_SAVE_YOURSELF**protocol , it is sent a message advising it to save its state. After a delay (specified by the resource *quitTimeout*), the application's connection to the X server is terminated. If the application observes neither of these protocols, its connection to the X server is terminated.

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

This function lowers a primary window to the bottom of the global window stack (where it obscures no other window) and lowers the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive. The *client* argument indicates the name or class of a client to lower. The name or class of a client appears in the **WM_CLASS** property on the client's top-level window. If the *client* argument is not specified, the context that the function was invoked in indicates the window or icon to lower. Specifying *within* lowers the secondary window within the family (staying above the parent) but does not lower the client family in the global window stack. Specifying *freeFamily* lowers the window to the bottom of the global windows stack from its local family stack.

f.maximize

This function causes a client window to be displayed with its maximum size. Refer to the *maximumClientSize*, *maximumMaximumSize*, and *limitResize* resources in **mwm**(1).

f.menu *menu_name*
This function associates a cascading (pull-right) menu with a menu pane entry or a menu with a button or key binding. The *menu_name* function argument identifies the menu to be used.

f.minimize
This function causes a client window to be minimized (iconified). When a window is minimized with no icon box in use, and if the *lowerOnIconify* resource has the value True (the default), the icon is placed on the bottom of the window stack (such that it obscures no other window). If an icon box is used, then the client's icon changes to its iconified form inside the icon box. Secondary windows (that is, transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move This function initiates an interactive move of a client window.

f.next_cmap
This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [*icon* | *window* | *transient*]
This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if **keyboardFocusPolicy** is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, then transient (secondary) windows are traversed (otherwise, if only *window* is specified, traversal is done only to the last focused window in a transient group). If an *icon* function argument is specified, then the function applies only to icons. If a *window* function argument is specified, then the function applies only to windows.

f.nop This function does nothing.

f.normalize
This function causes a client window to be displayed with its normal size. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.normalize_and_raise
This function causes a client window to be displayed with its normal size and raised to the top of the window stack. Secondary windows (that is, transient windows) are placed in their normal state along with their associated primary window.

f.pack_icons
This function is used to relayout icons (based on the layout policy being used) on the root window or in the icon box. In general this causes icons to be "packed" into the icon grid.

f.pass_keys
This function is used to enable/disable (toggle) processing of key bindings for window manager functions. When it disables key binding processing all keys are passed on to the window with the keyboard input focus and no window manager functions are invoked. If the **f.pass_keys** function is invoked with a key binding to disable key binding processing the same key binding can be used to enable key binding processing.

f.post_wmenu
This function is used to post the window menu. If a key is used to post the window menu and a window menu button is present, the window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window. If no window menu button is present, the window menu is placed at the top-left corner of the client 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*]

This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as **f.nop** if *keyboardFocusPolicy* is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, then transient (secondary) windows are traversed (otherwise, if only *window* is specified, traversal is done only to the last focused window in a transient group). If an *icon* function argument is specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.quit_mwm

This function terminates mwm (but NOT the X window system).

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

This function raises a primary window to the top of the global window stack (where it is obscured by no other window) and raises the secondary window (transient window or dialog box) within the client family. The arguments to this function are mutually exclusive. The *client* argument indicates the name or class of a client to lower. If the *client* is not specified, the context that the function was invoked in indicates the window or icon to lower. Specifying *within* raises the secondary window within the family but does not raise the client family in the global window stack. Specifying *freeFamily* raises the window to the top of its local family stack and raises the family to the top of the global window stack.

f.raise_lower [*within* | *freeFamily*]

This function raises a primary window to the top of the global window stack if it is partially obscured by another window; otherwise, it lowers the window to the bottom of the window stack. The arguments to this function are mutually exclusive. Specifying *within* raises a secondary window within the family (staying above the parent window), if it is partially obscured by another window in the application's family; otherwise, it lowers the window to the bottom of the family stack. It has no effect on the global window stacking order. Specifying *freeFamily* raises the window to the top of its local family stack, if obscured by another window, and raises the family to the top of the global window stack; otherwise, it lowers the window to the bottom of its local family stack and lowers the family to the bottom of the global window stack.

f.refresh This function causes all windows to be redrawn.

f.refresh_win

This function causes a client window to be redrawn.

f.resize This function initiates an interactive resize of a client window.

f.restore This function restores the previous state of an icon's associated window. If a maximized window is iconified, then **f.restore** restores it to its maximized state. If a normal window is iconified, then **f.restore** restores it to its normalized state.

f.restore_and_raise

This function restores the previous state of an icon's associated window and raises the window to the top of the window stack. If a maximized window is iconified, then **f.restore_and_raise** restores it to its maximized state and raises it to the top of the window stack. If a normal window is iconified, then **f.restore_and_raise** restores it to its normalized state and raises it to the top of the window stack.

f.restart This function causes mwm to be restarted (effectively terminated and re-executed). Restart is necessary for **mwm** to incorporate changes in both the **mwmrc** file and X resources.

f.screen [*next* | *prev* | *back* | *screen_number*]

This function causes the pointer to be warp to a specific screen number or to the *next*, *previous*, or last visited (*back*) screen. The arguments to this function are mutually exclusive. The *screen_number* argument indicates the screen number that the pointer is to be warped. Screens

are numbered starting from screen 0. Specifying *next* cause the pointer to warp to the next managed screen (skipping over any unmanaged screens). Specifying *prev* cause the pointer to warp to the previous managed screen (skipping over any unmanaged screens). Specifying *back* cause the pointer to warp to the last visited screen.

f.send_msg *message_number*
This function sends an **XClientMessageEvent** of type **_MOTIF_WM_MESSAGES** with *message_type* set to *message_number*. The client message is sent only if *message_number* is included in the client's **_MOTIF_WM_MESSAGES** property. A menu item label is grayed out if the menu item is used to do **f.send_msg** of a message that is not included in the client's **_MOTIF_WM_MESSAGES** property.

f.separator
This function causes a menu separator to be put in the menu pane at the specified location (the label is ignored).

f.set_behavior
This function causes the window manager to restart with the default behavior (if a custom behavior is configured) or a custom behavior (if a default behavior is configured). By default this is bound to *Shift Ctrl Alt <Key>!*.

f.title This function inserts a title in the menu pane at the specified location.

f.version This function causes the window manager to display its release version in a dialog box.

Function Constraints

Each function may be constrained as to which resource types can specify the function (for example, menu pane) and also what context the function can be used in (for example, the function is done to the selected client window). Function contexts are:

root No client window or icon has been selected as an object for the function.

window A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the window is in its normalized state (for example, **f.maximize**) or its maximized state (for example, **f.normalize**).

icon An icon has been selected as an object for the function.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply then the function is treated as **f.nop**. The following table indicates the resource types and function contexts in which window manager functions apply.

Function	Contexts	Resources
f.beep	root,icon,window	button,key,menu
f.circle_down	root,icon,window	button,key,menu
f.circle_up	root,icon,window	button,key,menu
f.exec	root,icon,window	button,key,menu
f.focus_color	root,icon,window	button,key,menu
f.focus_key	root,icon,window	button,key,menu
f.kill	icon,window	button,key,menu
f.lower	root,icon,window	button,key,menu
f.maximize	icon,window(normal)	button,key,menu
f.menu	root,icon,window	button,key,menu
f.minimize	window	button,key,menu
f.move	icon,window	button,key,menu
f.next_cmap	root,icon,window	button,key,menu
f.next_key	root,icon,window	button,key,menu
f.nop	root,icon,window	button,key,menu
f.normalize	icon,window(maximized)	button,key,menu
f.normalize_and_raise	icon,window	button,key,menu

f.pack_icons	root,icon,window	button,key,menu
f.pass_keys	root,icon,window	button,key,menu
f.post_wmenu	root,icon,window	button,key
f.prev_cmap	root,icon,window	button,key,menu
f.prev_key	root,icon,window	button,key,menu
f.quit_mwm	root	button,key,menu (root only)
f.raise	root,icon,window	button,key,menu
f.raise_lower	icon,window	button,key,menu
f.refresh	root,icon,window	button,key,menu
f.refresh_win	window	button,key,menu
f.resize	window	button,key,menu
f.restart	root	button,key,menu (root only)
f.restore	icon,window	button,key,menu
f.restore_and_raise	icon,window	button,key,menu
f.screen	root,icon,window	button,key,menu
f.send_msg	icon,window	button,key,menu
f.separator	root,icon,window	menu
f.set_behavior	root,icon,window	button,key,menu
f.title	root,icon,window	menu
f.version	root,icon,window	button,key,menu

WINDOW MANAGER EVENT SPECIFICATION

Events are indicated as part of the specifications for button and key binding sets, and menu panes. Button events have the following syntax:

button =~[**modifier_list**]<**button_event_name** >
modifier_list =~**modifier_name** { **modifier_name**}

The following table indicates the values that can be used for **modifier_name**. Note that [Alt] and [Meta] can be used interchangably on some hardware.

Modifier	Description
Ctrl	Control Key
Shift	Shift Key
Alt	Alt Key
Meta	Meta Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

Locking modifiers are ignored when processing button and key bindings. The following table lists keys that are interpreted as locking modifiers. The X server may map some of these symbols to the Mod1 - Mod5 modifier keys. These keys may or may not be available on your hardware: Key Symbol Caps Lock Shift Lock Kana Lock Num Lock Scroll Lock The following table indicates the values that can be used for **button_event_name**.

Button	Description
Btn1Down	Button 1 Press
Btn1Up	Button 1 Release
Btn1Click	Button 1 Press and Release
Btn1Click2	Button 1 Double Click
Btn2Down	Button 2 Press
Btn2Up	Button 2 Release
Btn2Click	Button 2 Press and Release

Btn2Click2	Button 2 Double Click
Btn3Down	Button 3 Press
Btn3Up	Button 3 Release
Btn3Click	Button 3 Press and Release
Btn3Click2	Button 3 Double Click
Btn4Down	Button 4 Press
Btn4Up	Button 4 Release
Btn4Click	Button 4 Press and Release
Btn4Click2	Button 4 Double Click
Btn5Down	Button 5 Press
Btn5Up	Button 5 Release
Btn5Click	Button 5 Press and Release
Btn5Click2	Button 5 Double Click

Key events that are used by the window manager for menu mnemonics and for binding to window manager functions are single key presses; key releases are ignored. Key events have the following syntax:

key =~[**modifier_list**] <**Key**>**key_name**
modifier_list =~**modifier_name** { **modifier_name**}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). Modifiers for keys are the same as those that apply to buttons. The **key_name** is an X11 keysym name. Keysym names can be found in the **keysymdef.h** file (remove the **XK_** prefix).

BUTTON BINDINGS

The **buttonBindings** resource value is the name of a set of button bindings that are used to configure window manager behavior. A window manager function can be done when a button press occurs with the pointer over a framed client window, an icon or the root window. The context for indicating where the button press applies is also the context for invoking the window manager function when the button press is done (significant for functions that are context sensitive). The button binding syntax is

Buttons **bindings_set_name**
{
button context function
button context function
. . .
button context function
}

The syntax for the **context** specification is: **context** = **object**[| **context**] **object** = *root* | *icon* | *window* | *title* | *frame* | *border* | *app* The context specification indicates where the pointer must be for the button binding to be effective. For example, a context of *window* indicates that the pointer must be over a client window or window management frame for the button binding to be effective. The *frame* context is for the window management frame around a client window (including the border and titlebar), the *border* context is for the border part of the window management frame (not including the titlebar), the *title* context is for the title area of the window management frame, and the *app* context is for the application window (not including the window management frame). If an **f.nop** function is specified for a button binding, the button binding is not done.

KEY BINDINGS

The **keyBindings** resource value is the name of a set of key bindings that are used to configure window manager behavior. A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings. The key binding syntax is:

Keys **bindings_set_name**
{

key context function
key context function
. . .
key context function
}

If an **f.nop** function is specified for a key binding, the key binding is not done. If an **f.post_wmenu** or **f.menu** function is bound to a key, **mwm** automatically uses the same key for removing the menu from the screen after it has been popped up. The **context** specification syntax is the same as for button bindings with one addition. The context *ifkey* may be specified for binding keys that may not be available on all displays. If the key is not available and if *ifkey* is in the context, then reporting of the error message to the error log is suppressed. This feature is useful for networked, heterogeneous environments. For key bindings, the *frame*, *title*, *border*, and *app* contexts are equivalent to the *window* context. The context for a key event is the window or icon that has the keyboard input focus (*root* if no window or icon has the keyboard input focus).

MENU PANES

Menus can be popped up using the **f.post_wmenu** and **f.menu** window manager functions. The context for window manager functions that are done from a menu is *root*, *icon* or *window* depending on how the menu was popped up. In the case of the *window* menu or menus popped up with a key binding, the location of the keyboard input focus indicates the context. For menus popped up using a button binding, the context of the button binding is the context of the menu. The menu pane specification syntax is:

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

Each line in the *Menu* specification identifies the label for a menu item and the function to be done if the menu item is selected. Optionally a menu button mnemonic and a menu button keyboard accelerator may be specified. Mnemonics are functional only when the menu is posted and keyboard traversal applies. The **label** may be a string or a bitmap file. The label specification has the following syntax:

label = text | bitmap_file
bitmap_file = @file_name
text = quoted_item | unquoted_item

The string encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the **f.nop** function or an invalid function or a function that doesn't apply in the current context. A **mnemonic** specification has the following syntax:

mnemonic = _ character

The first matching **character** in the label is underlined. If there is no matching **character** in the label, no mnemonic is registered with the window manager for that label. Although the **character** must exactly match a character in the label, the mnemonic does not execute if any modifier (such as Shift) is pressed with the character key. The **accelerator** specification is a key event specification with the same syntax as is used for key bindings to window manager functions.

INCLUDING FILES

You may include other files into your mwmrc file by using the *include* construct. For example,

```
INCLUDE
{
    /usr/local/shared/mwm.menus
    /home/kmt/personal/my.bindings
}
```

causes the files named to be read in and interpreted in order as an additional part of the mwmrc file. *Include* is a top-level construct. It cannot be nested inside another construct.

WARNINGS

Errors that occur during the processing of the resource description file are recorded in: **$HOME/.mwm/errorlog**. Be sure to check this file if the appearance or behavior of **mwm** is not what you expect.

FILES

$HOME/$LANG/.mwmrc
$HOME/.mwmrc
/usr/X11R6/lib/X11/$LANG/system.mwmrc
/usr/X11R6/lib/X11/system.mwmrc

RELATED INFORMATION

mwm(1), **X**(1).