NAME

Fvwm − F? Virtual Window Manager for X11

SYNOPSIS

fvwm [**−c** *config−command*] [**−d** *displayname*] [**−f** *config−file*] [**−r**] [**−s** [*screen_num*]] [**−V**]
[**−C** *visual−class* | **−I** *visual−id*] [**−l** *colors* [**−L**] [**−A**] [**−S**] [**−P**]] [**−D**] [**−h**] [**−i** *client−id*]
[**−F** *state−file*] [**−−debug−stack−ring**] [**−blackout**]

DESCRIPTION

Fvwm is a window manager for X11. It is designed to minimize memory consumption, provide a 3D look to window frames, and a virtual desktop.

Note that there are several window managers around that have "fvwm" in their name. In the past, version 2.x of fvwm was commonly called fvwm2 to distinguish it from the former version 1.x (fvwm or even fvwm1). Since version 1.x has been replaced by version 2.x a long time ago we simply call version 2.x and all versions to come, fvwm, throughout this document, and the executable program is named fvwm. There is an fvwm offspring called fvwm95, it is mostly a patched version of fvwm−2.0.43. The main goal of fvwm95 was to supply a Windows 95 like look and feel. Since then, fvwm has been greatly enhanced and practically all fvwm95 features can be achieved by fvwm.

Fvwm provides both a large *virtual desktop* and *multiple disjoint desktops* which can be used separately or together. The virtual desktop allows you to pretend that your video screen is really quite large, and you can scroll around within the desktop. The multiple disjoint desktops allow you to pretend that you really have several screens to work at, but each screen is completely unrelated to the others.

Fvwm provides *keyboard accelerators* that allow you to perform most window manager functions, including moving and resizing windows and operating the menus, using keyboard shortcuts.

Fvwm has also overcome the distinction between configuration commands and action commands that most window managers make. Configuration commands typically set fonts, colors, menu contents, and key and mouse function bindings, while action commands do things like raise and lower windows. Fvwm makes no such distinction and allows anything to be changed at any time.

Other noteworthy differences between fvwm and other X11 window managers are the introduction of the *SloppyFocus* and *NeverFocus* focus methods. Focus policy can be separately specified for different window groups. Windows using *SloppyFocus* acquire focus when the pointer moves into them and retain focus until some other window acquires it. Such windows do not lose focus when the pointer moves into the root window. The*Ne verFocus* policy is provided for use with windows into which one never types (e.g. xclock, oclock, xbiff, xeyes, tuxeyes) − for example, if a SloppyFocus terminal window has focus, moving the pointer over a NeverFocus decoration window does not deprive the terminal of focus.

OPTIONS

These are the command line options that are recognized by fvwm:

−i | **−−clientid** *id*
This option is used when fvwm is started by a session manager. Should not be used by a user.

−c | **−−cmd** *config−command*
Causes fvwm to use *config−command* instead of '**Read** *config*' (or '**Read** *.fvwm2rc*') as its initialization command. (Note that up to 10 **−f** and **−c** parameters can be given, and they are executed in the order specified.)

Any module started by command line arguments is assumed to be a module that sends back config commands. All command line modules have to quit before fvwm proceeds on to the StartFunction and setting border decorations and styles. There is a potential deadlock if you start a module other than **FvwmCpp**/**FvwmM4**/**FvwmPerl** but there is a timeout so fvwm eventually gets going.

As an example, starting the pager this way hangs fvwm until the timeout, but the following should work well:

fvwm −c "**AddToFunc** StartFunction I **Module FvwmPager**"

−d | **−−display** *displayname*
Manage the display called *displayname* instead of the name obtained from the environment variable *$DISPLAY*.

−D | **−−debug**
Puts X transactions in synchronous mode, which dramatically slows things down, but guarantees that fvwm's internal error messages are correct. Also causes fvwm to output debug messages while running.

−f *config−file*
Causes fvwm to read *config−file* instead of *~/.fvwm/config* as its initialization file. This is equivalent to −c '**Read** *config−file*'.

−h | **−−help**
A short usage description is printed.

−r | **−−replace**
Try to take over from a previously running wm. This does not work unless the other wm is ICCCM2 2.0 compliant.

−F | **−−restore** *state−file*
This option is used when fvwm is started by a session manager. Should not be used by a user.

−s | **−−single−screen** [*screen_num*]
On a multi−screen display, run fvwm only on the screen named in the *$DISPLAY* environment variable or provided through the **−d** option. The optional argument *screen_num* should be positive or null and override the screen number. Normally, fvwm attempts to start up on all screens of a multi−screen display.

−V | **−−version**
Prints the version of fvwm to *stderr*. Also prints an information about the compiled in support for readline, rplay, stroke, xpm, png, svg, GNOME hints, EWMH hints, session management, bidirectional text, multibyte characters, xinerama and Xft aa font rendering.

−C | **−−visual** *visual−class*
Causes fvwm to use *visual−class* for the window borders and menus. *visual−class* can be "StaticGray", "GrayScale", "StaticColor", "PseudoColor", "TrueColor" or "DirectColor".

−I | **−−visualid** *id*
Causes fvwm to use *id* as the visual id for the window borders and menus. *id* can be specified as N for decimal or 0xN for hexadecimal. See man page of xdpyinfo for a list of supported visuals.

−l | **−−color−limit** *limit*
Specifies a *limit* on the colors used in image, gradient and possibly simple colors used by fvwm. In fact, fvwm (and all the modules) uses a palette with at most *limit* colors. This option is only useful with screens that display 256 colors (or less) with a dynamic visual (PseudoColor, GrayScale or DirectColor). The default depends on your X server and how you run fvwm. In most case this default is reasonable. The **−l** option should be used only if you encounter problems with colors. By default, fvwm tries to detect large pre−allocated palettes. If such a palette is detected fvwm uses it and a priori the **−l** must not be used. Moreover, in this case the **−A** and **−S** options are forced. Note that XFree−4.2 pre−allocates 244 colors (if you use a driver with Render support) leaving only a few free colors. This may lead to some color problems (and nothing can be done). XFree−4.3 or better pre−allocate only 85 colors. If no pre−allocated palette is auto detected the defaults are as follow:

Display depth 8 (256 colors)

PseudoColor: 68 (4x4x4 color cube + 4 grey)
GrayScale: 64 regular grey
DirectColor: 32 (3x3x3 color cube + 5 grey)

Display depth 4 (16 colors)

PseudoColor: 10 (2x2x2 color cube + 2 grey)
GrayScale: 8 regular grey
DirectColor: 10 (2x2x2 color cube + 2 grey)

These defaults may change before version 2.6. Note that if you use a private color map (i.e., fvwm is started with the **−C** or the **−I** options), then other defaults are used.

Now what to do if you encounter problems with colors? The first thing to do is to check if you really cannot run your X server with depth 15, 16 or better. Check your X server documentation. Note that some hardware can support two different depths on the same screen (typically depth 8 and depth 24). If depth 8 is the default, you can force fvwm to use the best depth by using the **−C** option with *TrueColor* as argument. So now we assume that you are forced to run in depth 8 with a dynamic visual because your hardware/driver cannot do better or because you need to use an application which needs to run under this mode (e.g., because this application needs read−write colors). What it should be understand is that you have only 256 colors and that all the applications which use the default color map must share these colors. The main problem is that there are applications which use a lot or even all the colors. If you use such application you may have no more free colors and some applications (which used only a few colors) may fail to start or are unusable. There are three things that can be done (and fvwm does not really play a particular role, all applications are concerned). The first is to run the applications which waste your (default) color map with a private color map. For example, run netscape with the −install option, run KDE or QT applications with the −−cmap option, use the **−C** option for fvwm. The disadvantage of this method is that it is visually disturbing (see the **ColormapFocus** command for a better control of the color maps switching). The second method is to limit the number of colors that the applications use. Again, some applications have options to specify a given color limit. With fvwm you may try various values, 61 (a special "visual" palette), 56 (a 4x4x3 color cube plus 6 grey), 29 (a 3x3x3 color cube plus 2 grey), 10 or 9. Also, you may use the **−L** option. However, limiting the number of colors is not the definitive solution. The definitive solution is to try cause applications which use a lot of colors use the same colors. This is a difficult task as there are no formal standards for this goal. However, some toolkits as QT and GTK use color cubes as palettes. So, the idea is to configure your applications/toolkits to all use the same color cube. Moreover, you can use the colors in this color cube in your X resources configuration files and/or as arguments to colors options. Fvwm can use any color cube of the form RxGxB with 2 <= R <= 6, R = G, R−1 =< B <= R and B >= 2. To get an RxGxB color cube give an argument to **−l** an integer c >= R*G*B and < (R+1)*(G+1)*B if B=R and < R*G*(B+1) if B < R (and different from 61). If c > R*G*B, then some grey may be added to the color cube. You can use the **PrintInfo** *Colors* [*1*] command to get information on your fvwm colors setting. In particular, this command prints the palette used by fvwm in rgb format (the last integer gives the number of times fvwm has allocated the colors).

−L | **−−strict−color−limit**
If the screen displays 256 colors (or less) and has a dynamic visual, causes fvwm to use its palette for all the colors. By default, the palette is used only for images and gradients.

−P | **−−visual−palette**
If the screen displays 256 colors (or less) and has a dynamic visual, this option causes fvwm to use a palette designed for limiting the "visual" color distance between the points of the palette. Moreover, for better color sharing, if possible colors with a name in the X rgb data base are used for defining the colors (with the hope that applications and images prefer to use named colors). If the **−l** option is not used this palette has 61 colors. This palette is also automatically selected if 61 or 9 is used as argument to the **−l** option.

−A | **−−allocate−palette**
If the screen displays 256 colors (or less) and has a dynamic visual this option causes fvwm to allocate all the colors of its palette at start up for reserving these colors for future use. This option forces the **−static−palette** option. By default, fvwm allocates (reserves) a color in its palette only if it needs this color.

−S | **−−static−palette**
If the screen displays 256 colors (or less) and has a dynamic visual this option causes fvwm to never free the colors in its palette. By default, when fvwm does not need a color any more it frees this color so that a new color can be used. This option may speed up image loading and save a few bits of memory.

−blackout
This option is provided for backward compatibility only. Blacking out the screen during startup is not necessary (and doesn't work) anymore. This option will be removed in the future.

−−debug−stack−ring
Enables stack ring debugging. This option is only intended for internal debugging and should only be used by developers.

ANATOMY OF A WINDOW

Fvwm puts a decorative border around most windows. This border consists of a bar on each side and a small L−shaped section on each corner. There is an additional top bar called the title−bar which is used to display the name of the window. In addition, there are up to 10 title−bar buttons. The top, side, and bottom bars are collectively known as the side−bars. The corner pieces are called the frame.

With the built−in minimal configuration, dragging mouse button 1 in the frame or side−bars begins a resize operation on the window. Dragging mouse button 2 in the frame or side−bars begins a move operation. There are raise/lower operations bound to a single clicking on borders. Similarly for the window title.

Up to ten title−bar buttons may exist. Their use is completely user definable. One popular configuration uses one button on the left that is used to bring up a list of window options and two buttons on the right used to iconify and maximize the window. Another popular configuration adds a close button to the right. The number of title−bar buttons used depends on which ones have mouse actions bound to them. See the **Mouse** command.

THE VIRTUAL DESKTOP

Fvwm provides multiple virtual desktops for users who wish to use them. The screen is a viewport onto a *desktop* which may be larger than the screen. Several distinct desktops can be accessed (concept: one desktop for each project, or one desktop for each application, when view applications are distinct). Since each desktop can be larger than the physical screen, divided into m by n *pages* which are each the size of the physical screen, windows which are larger than the screen or large groups of related windows can easily be viewed.

The (m by n) size (i.e. number of pages) of the virtual desktops can be changed any time, by using the **DesktopSize** command. All virtual desktops must be (are) the same size. The total number of distinct desktops does not need to be specified, but is limited to approximately 4 billion total. All windows on a range of desktops can be viewed in the **FvwmPager**, a miniature view of the desktops. The pager is an accessory program, called a module, which is not essential for the window manager to operate. Windows may also be listed using the **WindowList** command or the **FvwmIconMan** module.

Fvwm keeps the windows on the desktop in a layered stacking order; a window in a lower layer never obscures a window in a higher layer. The layer of a window can be changed by using the **Layer** command. The concept of layers is a generalization of the *StaysOnTop* flag of older fvwm versions. The*StaysOnTop* and *StaysPut* **Style** options are now implemented by putting the windows in suitable layers and the previously missing *StaysOnBottom* **Style** option has been added.

Sticky windows are windows which transcend the virtual desktop by "Sticking to the screen's glass". They always stay put on the screen. This is convenient for things like clocks and xbiffs, so you only need to run one such gadget and it always stays with you. Icons can also be made to stick to the glass, if desired.

Window geometries are specified relative to the current viewport. That is:

```
xterm −geometry +0+0
```

creates a window in the upper left hand corner of the visible portion of the screen. It is permissible to specify geometries which place windows on the virtual desktop, but off the screen. For example, if the

visible screen is 1000 by 1000 pixels, and the desktop size is 3x3, and the current viewport is at the upper left hand corner of the desktop, invoking:

xterm −geometry +1000+1000

places a window just off of the lower right hand corner of the screen. It can be found by moving the mouse to the lower right hand corner of the screen and waiting for it to scroll into view. A geometry specified as something like:

xterm −geometry −5−5

places the window's lower right hand corner 5 pixels from the lower right corner of the visible portion of the screen. Not all applications support window geometries with negative offsets. Some applications place the window's upper right hand corner 5 pixels above and to the left of the upper left hand corner of the screen; others may do just plain bizarre things.

There are several ways to cause a window to map onto a desktop or page other than the currently active one. The geometry technique mentioned above (specifying x,y coordinates larger than the physical screen size), however, suffers from the limitation of being interpreted relative to the current viewport: the window may not consistently appear on a specific page, unless you always invoke the application from the same page.

A better way to place windows on a different page, screen or desk from the currently mapped viewport is to use the *StartsOnPage* or *StartsOnScreen* style specification (the successors to the older *StartsOnDesk* style) in your *config* file. The placement is consistent: it does not depend on your current location on the virtual desktop.

Some applications that understand standard Xt command line arguments and X resources, like xterm and xfontsel, allow the user to specify the start−up desk or page on the command line:

xterm −xrm "*Desk:1"

starts an xterm on desk number 1;

xterm −xrm "*Page:3 2 1"

starts an xterm two pages to the right and one down from the upper left hand page of desk number 3. Not all applications understand the use of these options, however. You could achieve the same results with the following lines in your *.Xdefaults* file:

XTerm*Desk: 1

or

XTerm*Page: 3 2 1

USE ON MULTI−SCREEN DISPLAYS

If the **−s** command line argument is not given, fvwm automatically starts up on every screen on the specified display. After fvwm starts each screen is treated independently. Restarts of fvwm need to be performed separately on each screen. The use of

EdgeScroll 0 0

is strongly recommended for multi−screen displays. You may need to quit on each screen to quit from the X session completely. This is not to be confused with Xinerama support.

XINERAMA SUPPORT

Fvwm supports the Xinerama extension of newer X servers which is similar to multi head support (multiple screens) but allows one to move windows between screens. If Xinerama support has been compiled into fvwm, it is used whenever fvwm runs on an X server that supports and uses multiple screens via Xinerama. Without this option, the whole desktop is treated as one big screen. For example, menus might pop up right

between two screens. The *EdgeResistance* option of the **Style** command command allows for specifying an explicit resistance value for moving windows over the screen edge between two Xinerama screens. Xinerama support can be enabled or disabled on the fly or from the configuration file with the **Xinerama** command. Many modules and commands work nicely with Xinerama displays.

Whenever a geometry in the usual X format can be supplied, fvwm's Xinerama extension allows for specifying a screen in addition to the geometry (or even the screen alone). To do this, a '@' is added to the end of the geometry string followed by either the screen number or a letter. A number is taken as the number of the Xinerama screen to be used (as configured in the X server). The letter can be one of 'g' for the global screen (the rectangle that encloses all Xinerama screens), 'p' for the primary screen (see below), 'c' for the current screen (the one that currently contains the pointer). If the X server does not support Xinerama or only one screen is used, the screen bit is ignored.

Style * *IconBox* 64x300−0−0@p

Xinerama support can be configured to use a primary screen. Fvwm can be configured to place new windows and icons on this screen. The primary screen is screen 0 by default but can be changed with the **XineramaPrimaryScreen** command.

Xinerama support was designed to work out of the box with the same configuration file that would work on a single screen. It may not perform very well if the involved screens use different screen resolutions. In this situation, windows may get stuck in the portion of the whole desktop that belongs to neither screen. When this happens, the windows or icons can be retrieved with the command

All MoveToScreen

that can be entered in an **FvwmConsole** window or with **FvwmCommand**.

For multi−screen implementations other than Xinerama, such as Single Logical Screen, it is possible to simulate a Xinerama configuration if the total screen seen by fvwm is made up of equal sized monitors in a rectangular grid. The commands **XineramaSls**, **XineramaSlsSize** and **XineramaSlsScreens** are used to configure this feature.

INITIALIZATION

During initialization, fvwm searches for a configuration file which describes key and button bindings, and many other things. The format of these files is described later. Fvwm first searches for configuration files using the command

Read *config*

This looks for file *config* in *$FVWM_USERDIR* and *$FVWM_DATADIR* directories, as described in **Read**. If this fails more files are queried for backward compatibility. Here is the complete list of all file locations queried in the default installation (only the first found file is used):

$HOME/.fvwm/config
/usr/local/share/fvwm/config

$HOME/.fvwm/.fvwm2rc
$HOME/.fvwm2rc
/usr/local/share/fvwm/.fvwm2rc
/usr/local/share/fvwm/system.fvwm2rc
/etc/system.fvwm2rc

Please note, the last 5 locations are not guaranteed to be supported in the future.

If a configuration file is not found, the left mouse button, or Help or F1 keys on the root window bring up menus and forms that can create a starting configuration file.

Fvwm sets two environment variables which are inherited by its children. These are *$DISPLAY* which describes the display on which fvwm is running. *$DISPLAY* may be *unix:0.0* or *:0.0*, which doesn't work too well when passed through ssh to another machine, so *$HOSTDISPLAY* is set to a network−ready description of the display. *$HOSTDISPLAY* always uses the TCP/IP transport protocol (even for a local

connection) so *$DISPLAY* should be used for local connections, as it may use Unix−domain sockets, which are faster.

If you want to start some applications or modules with fvwm, you can simply put

Exec app

or

Module FvwmXxx

into your *config*, but it is not recommended; do this only if you know what you are doing. It is usually important to start applications or modules after the entire config is read, because it contains styles or module configurations which can affect window appearance and functionality.

The standard way to start applications or modules on fvwm's start up is to add them to an initialization function (usually **StartFunction** or **InitFunction**). This way they are only started after fvwm finishes to read and execute *config* file.

Fvwm has three special functions for initialization: **StartFunction**, which is executed on startups and restarts; **InitFunction** and **RestartFunction**, which are executed during initialization and restarts (respectively) just after StartFunction. These functions may be customized in a user's *config* file using the **AddToFunc** command (described later) to start up modules, xterms, or whatever you'd like to have started by fvwm.

Fvwm has also a special exit function: **ExitFunction**, executed when exiting or restarting before actually quitting. It could be used to explicitly kill modules, etc.

If fvwm is run under a session manager, functions **SessionInitFunction** and **SessionRestartFunction** are executed instead of InitFunction and RestartFunction. This helps to define the user's *config* file to be good for both running under a session manager and without it. Generally it is a bad idea to start xterms or other applications in "Session*" functions. Also someone can decide to start different modules while running under a session manager or not. For the similar purposes **SessionExitFunction** is used instead of ExitFunction.

DestroyFunc StartFunction
AddToFunc StartFunction
+ I **Module FvwmPager** * *
+ I **Module FvwmButtons**

DestroyFunc InitFunction
AddToFunc InitFunction
+ I **Module FvwmBanner**
+ I **Module FvwmIconMan**
+ I **Exec** xsetroot −solid cyan
+ I **Exec** xterm
+ I **Exec** netscape

DestroyFunc RestartFunction
AddToFunc RestartFunction
+ I **Module FvwmIconMan**

DestroyFunc SessionInitFunction
AddToFunc SessionInitFunction
+ I **Module FvwmBanner**

DestroyFunc SessionRestartFunction
AddToFunc SessionRestartFunction
+ I **Nop**

You do not need to define all special functions if some are empty. Also note, all these special functions may be emulated now using **StartFunction** and **ExitFunction,** like this:

DestroyFunc StartFunction
AddToFunc StartFunction
+ I **Test** (Init) **Module FvwmBanner**
+ I **Module FvwmPager** * *
+ I **Test** (Restart) **Beep**

DestroyFunc ExitFunction
AddToFunc ExitFunction
+ I **Test** (Quit) **Echo** Bye−bye
+ I **KillModule** MyBuggyModule
+ I **Test** (ToRestart) **Beep**

COMPILATION OPTIONS

Fvwm has a number of compile−time options. If you have trouble using a certain command or feature, check to see if support for it was included at compile time. Optional features are described in the *config.h* file that is generated during compilation.

ICONS AND IMAGES

Fvwm can load **.xbm, .xpm, .png** and **.svg** images. **XBM** images are monochrome. Fvwm can always display **XBM** files. **XPM** and **PNG** formats are color images. SVG is a vector graphics image format. Compile−time options determine whether fvwm can display **XPM**, **PNG** or **SVG** icons and images. See the *INSTALL.fvwm* file for more information.

The related **SHAPE** compile−time option can make fvwm display spiffy shaped icons.

SVG rendering options

SVG images are generated from (XML) text files. A really simple SVG file might look something like this:

```
<svg width="120" height="80">
        <rect fill="red"     width="40" height="40"  x="0"   y="0"  />
        <rect fill="lime"    width="40" height="40"  x="40"  y="0"  />
        <rect fill="blue"    width="40" height="40"  x="80"  y="0"  />
        <rect fill="cyan"    width="40" height="40"  x="0"   y="40" />
        <rect fill="magenta" width="40" height="40"  x="40"  y="40" />
        <rect fill="yellow"  width="40" height="40"  x="80"  y="40" />
</svg>
```

By default, SVG images are rendered as the image creator intended them to. But since SVG is a vector graphics format, the images can be rendered at any chosen size and rotation, e.g. making it possible to use the same icon file rendered at different sizes for the *Icon* and *MiniIcon* styles.

The rendering options are specified as a string appended to the SVG filename as follows:

image.svg:[!] [(1) *size*] [(2) *position*] [(3) *rotation*] [(4) *scale*] ...

(1) [−]*width*{x}[−]*height*
(2) {− | +}*xpos*{− | +}*ypos*
(3) @[−]*angle*
(4) {* | /}[−]*factor*[x | y]

The option string always starts with a colon (':') to separate it from the filename. An empty option string can skip this colon, but it might still be a good idea to include it to prevent ambiguity if the filename contains any colon.

filename_without_colon.svg
filename:with:colon.svg:

An exclamation point ('!') transposes the entire final image (including the rendering area), i.e. all the

horizontal and all the vertical coordinates are swapped with each other.

image.svg:!

width and *height* specifies the dimensions of the rendering area in pixels, i.e. the dimensions of the resulting image. The actual image is fitted to fill the entire rendering area.

image.svg:60x60

Use a *width* or *height* value of 0 to keep the aspect ratio.

image.svg:0x60
image.svg:60x0

A '−' before *width* mirrors the rendering area horizontally.

image.svg:−0x0

A '−' before *height* mirrors the rendering area vertically.

image.svg:0x−0

xpos and *ypos* specifies a translation of the image in pixels. A positive *xpos* value moves the image to the right. A positive *ypos* value moves it down. Moving it partially outside of the rendering area results in a cropped image.

image.svg:−30−0
image.svg:−0+10
image.svg:−30+10

angle specifies a rotation around the actual image center in degrees. This might result in a cropped image. A positive value rotates the image clockwise. Floating point values are recognized.

image.svg:@180
image.svg:@−90
image.svg:@30
image.svg:@57.3

factor specifes a scaling of the actual image (not the rendering area). Scaling it up results in a cropped image. Floating point values are recognized. Division by zero is ignored. If *factor* is directly followed by a 'x' or a 'y', the scaling is horizontal or vertical respectively. Otherwise the scaling is uniform.

image.svg:*2
image.svg:/2
image.svg:/3x
image.svg:/2y

Scaling down a translated or rotated image can prevent cropping.

image.svg:@30*0.6

Repeated usage of translation, rotation, and scaling is allowed. Translation and rotation are additive. Scaling is multiplicative.

image.svg:*2/3
image.svg:/3x/2y

When combining affine transformations, the scaling is always done first, then the rotation, and finally the translation.

image.svg:−30+10@30/3x/2y

Use a negative scale *factor* to mirror the actual image.

image.svg:−30+10@30/−3x/2y

Mirroring of the rendering area is done after any scaling, rotation or translation of the image.

image.svg:−0x0−30+10@30/3x/2y

Transposing is done last of all, after everything else.

image.svg:!−0x0−30+10@30/3x/2y

MODULES

A module is a separate program which runs as a separate Unix process but transmits commands to fvwm to execute. Users can write their own modules to do any weird or bizarre manipulations without bloating or affecting the integrity of fvwm itself.

Modules must be spawned by fvwm so that it can set up two pipes for fvwm and the module to communicate with. The pipes are already open for the module when it starts and the file descriptors for the pipes are provided as command line arguments.

Modules can be spawned by fvwm at any time during the X session by use of the **Module** command. Modules can exist for the duration of the X session, or can perform a single task and exit. If the module is still active when fvwm is told to quit, then fvwm closes the communication pipes and waits to receive a SIGCHLD from the module, indicating that it has detected the pipe closure and has exited. If modules fail to detect the pipe closure fvwm exits after approximately 30 seconds anyway. The number of simultaneously executing modules is limited by the operating system's maximum number of simultaneously open files, usually between 60 and 256.

Modules simply transmit commands to the fvwm command engine. Commands are formatted just as in the case of a mouse binding in the *config* setup file. Certain auxiliary information is also transmitted, as in the sample module **FvwmButtons**.

Please refer to the **Module Commands** section for details.

ICCCM COMPLIANCE

Fvwm attempts to be ICCCM 2.0 compliant. Check *http://tronche.com/gui/x/icccm/* for more info. In addition, ICCCM states that it should be possible for applications to receive any keystroke, which is not consistent with the keyboard shortcut approach used in fvwm and most other window managers. In particular you cannot have the same keyboard shortcuts working with your fvwm and another fvwm running within Xnest (a nested X server running in a window). The same problem exists with mouse bindings.

The ICCCM states that windows possessing the property

WM_HINTS(WM_HINTS):
  Client accepts input or input focus: False

should not be given the keyboard input focus by the window manager. These windows can take the input focus by themselves, however. A number of applications set this property, and yet expect the window manager to give them the keyboard focus anyway, so fvwm provides a window style, *Lenience*, which allows fvwm to overlook this ICCCM rule. Even with this window style it is not guaranteed that the application accepts focus.

The differences between ICCCM 1.1 and 2.0 include the ability to take over from a running ICCCM 2.0 compliant window manager; thus

fvwm; vi ~/.fvwm/config; fvwm −replace

resembles the **Restart** command. It is not exactly the same, since killing the previously running wm may

terminate your X session, if the wm was started as the last client in your *.Xclients* or *.Xsession* file.

Further additions are support for client−side colormap installation (see the ICCCM for details) and the urgency hint. Clients can set this hint in the WM_HINTS property of their window and expect the window manager to attract the user's attention to the window. Fvwm has two re−definable functions for this purpose, "UrgencyFunc" and "UrgencyDoneFunc", which are executed when the flag is set/cleared. Their default definitions are:

AddToFunc UrgencyFunc
+ I **Iconify** off
+ I **FlipFocus**
+ I **Raise**
+ I **WarpToWindow !raise** 5p 5p
AddToFunc UrgencyDoneFunc
+ I **Nop**

GNOME COMPLIANCE

Fvwm attempts to be GNOME (version 1) compliant. Check *http://www.gnome.org* for what that may mean. To disable GNOME hints for some or all windows, the *GNOMEIgnoreHints* style can be used.

EXTENDED WINDOW MANAGER HINTS

Fvwm attempts to respect the extended window manager hints (ewmh or EWMH for short) specification: *http://www.freedesktop.org/wiki/Standards_2fwm_2dspec* and some extensions of this specification. This allows fvwm to work with KDE version >= 2, GNOME version 2 and other applications which respect this specification (any application based on *GTK+* version 2). Applications which respect this specification are called ewmh compliant applications.

This support is configurable with styles and commands. These styles and commands have EWMH as the prefix (so you can find them easily in this man page).

There is a new Context 'D' for the **Key**, **PointerKey**, **Mouse** and **Stroke** commands. This context is for desktop applications (such as kdesktop and Nautilus desktop).

When a compliant taskbar asks fvwm to activate a window (typically when you click on a button which represents a window in such a taskbar), then fvwm calls the complex function **EWMHActivateWindowFunc** which by default is Iconify Off, Focus and Raise. You can redefine this function. For example:

DestroyFunc EWMHActivateWindowFunc
AddToFunc EWMHActivateWindowFunc I **Iconify** Off
+ I **Focus**
+ I **Raise**
+ I **WarpToWindow** 50 50

additionally warps the pointer to the center of the window.

The EWMH specification introduces the notion of Working Area. Without ewmh support the Working Area is the full visible screen (or all your screens if you have a multi head setup and you use Xinerama). However, compliant applications (such as a panel) can ask to reserve space at the edge of the screen. If this is the case, the Working Area is your full visible screen minus these reserved spaces. If a panel can be hidden by clicking on a button the Working Area does not change (as you can unhide the panel at any time), but the Dynamic Working Area is updated: the space reserved by the panel is removed (and added again if you pop up the panel). The Dynamic Working Area may be used when fvwm places or maximizes a window. To know if an application reserves space you can type "xprop | grep _NET_WM_STRUT" in a terminal and select the application. If four numbers appear then these numbers define the reserved space as explained in the **EwmhBaseStruts** command.

MWM COMPATIBILITY

Fvwm provides options to emulate Motif Window Manager (Mwm) as well as possible. Please refer to the **Emulate** command as well as to the Mwm specific options of the **Style** and **MenuStyle** commands for

details.

OPEN LOOK AND XVIEW COMPATIBILITY

Fvwm supports all the Open Look decoration hints (except pushpins). Should you use any such application, please add the following line to your config:

Style * *OLDecor*

Most (perhaps all) Open Look applications have a strange notion of keyboard focus handling. Although a lot of work went into fvwm to work well with these, you may still encounter problems. It is recommended to use the *NeverFocus* focus policy and the *Lenience* style for all such applications (the windows still get the focus):

Style <application name> *NeverFocus*, *Lenience*

But in case you can not live with that focus policy, you can try using one of the other focus policies in combination with the *Lenience* style:

Style <application name> *MouseFocus*, *Lenience*
Style <application name> *SloppyFocus*, *Lenience*
Style <application name> *ClickToFocus*, *Lenience*

M4 PREPROCESSING

M4 pre−processing is handled by a module in fvwm. To get more details, try man **FvwmM4**. In short, if you want fvwm to parse your files with m4, then replace the command **Read** with **FvwmM4** in your *~/.fvwm/config* file (if it appears at all), and start fvwm with the command

fvwm −cmd "**FvwmM4** config"

CPP PREPROCESSING

Cpp is the C−language pre−processor. fvwm offers cpp processing which mirrors the m4 pre−processing. To find out about it, re−read the **M4** section, but replace "m4" with "cpp".

CONFIGURATION

Configuration Files

The configuration file is used to describe mouse and button bindings, colors, the virtual display size, and related items. The initialization configuration file is typically called *config* (or *.fvwm2rc*). By using the **Read** command, it is easy to read in new configuration files as you go.

Lines beginning with '#' are ignored by fvwm. Lines starting with '*' are expected to contain module configuration commands (rather than configuration commands for fvwm itself). Like in shell scripts embedded newlines in a configuration file line can be quoted by preceding them with a backslash. All lines linked in this fashion are treated as a single line. The newline itself is ignored.

Fvwm makes no distinction between configuration commands and action commands, so anything mentioned in the fvwm commands section can be placed on a line by itself for fvwm to execute as it reads the configuration file, or it can be placed as an executable command in a menu or bound to a mouse button or a keyboard key. It is left as an exercise for the user to decide which function make sense for initialization and which ones make sense for run−time.

Supplied Configuration

A sample configuration file, is supplied with the fvwm distribution. It is well commented and can be used as a source of examples for fvwm configuration. It may be copied from */usr/local/share/fvwm/config* file.

Alternatively, the built−in menu (accessible when no configuration file is found) has options to create an initial config file for the user.

FONTS

Font names and font loading

The fonts used for the text of a window title, icon titles, menus and geometry window can be specified by using the Font and IconFont **Style**, the Font **MenuStyle** and the **DefaultFont** commands. Also, all the

Modules which use text have configuration command(s) to specify font(s). All these styles and commands take a font name as an argument. This section explains what is a font name for fvwm and which fonts fvwm loads.

First, you can use what we can call a usual font name, for example,

−adobe−courier−bold−r−normal−−10−100−75−75−m−60−ISO8859−1
−adobe−courier−bold−r−normal−−10−*
−*−fixed−medium−o−normal−−14−*−ISO8859−15

That is, you can use an X Logical Font Description (XLFD for short). Then the "first" font which matches the description is loaded and used. This "first" font depends of your font path and also of your locale. Fonts which match the locale charset are loaded in priority order. For example with

−adobe−courier−bold−r−normal−−10−*

if the locale charset is ISO8859−1, then fvwm tries to load a font which matches

−adobe−courier−bold−r−normal−−10−*−ISO8859−1

with the locale charset ISO8859−15 fvwm tries to load

−adobe−courier−bold−r−normal−−10−*−ISO8859−15.

A font name can be given as an extended XLFD. This is a comma separated list of (simple) XLFD font names, for example:

−adobe−courier−bold−r−normal−−14−*,−*−courier−medium−r−normal−−14−*

Each simple font name is tried until a matching font with the locale charset is found and if this fails each simple font name is tried without constraint on the charset.

More details on the XLFD can be found in the X manual page, the X Logical Font Description Conventions document (called xlfd) and the XLoadFont and XCreateFontSet manual pages. Some useful font utilities are: xlsfonts, xfontsel, xfd and xset.

If you have Xft support you can specify an Xft font name (description) of a true type (or Type1) font prefixed by "xft:", for example:

"xft:Luxi Mono"
"xft:Luxi Mono:Medium:Roman:size=14:encoding=iso8859−1"

The "first" font which matches the description is loaded. This first font depends on the XftConfig configuration file with Xft1 and on the /etc/fonts/fonts.conf file with Xft2. One may read the Xft manual page and the fontconfig man page with Xft2. The first string which follows "xft:" is always considered as the family. With the second example Luxi Mono is the Family (Other XFree TTF families: "Luxi Serif", "Luxi Sans"), Medium is the Weight (other possible weights: Light, DemiBold, Bold, Black), Roman is the slant or the style (other possibilities: Regular, Oblique, Italic) size specifies the point size (for a pixel size use pixelsize=), encoding allows for enforce a charset (iso8859−1 or iso10646−1 only; if no encoding is given the locale charset is assumed). An important parameter is "minspace=bool" where bool is True or False. If bool is False (the default?) Xft gives a greater font height to fvwm than if bool is True. This may modify text placement, icon and window title height, line spacing in menus and **FvwmIdent**, button height in some fvwm modules ...etc. With a LCD monitor you may try to add "rgba=mode" where mode is either rgb, bgr, vrgb or vbgr to enable subpixel rendering. The best mode depends on the way your LCD cells are arranged. You can pass other specifications in between ":", as "foundry=foundry_name", "spacing=type" where type can be monospace, proportional or charcell, "charwidth=integer", "charheight=integer" or "antialias=bool" where bool is True or False. It seems that these parameters are not always taken in account.

To determine which Xft fonts are really loaded you can export XFT_DEBUG=1 before starting fvwm and

take a look to the error log. With Xft2 you may use fc−list to list the available fonts. Anyway, Xft support is experimental (from the X and the fvwm point of view) and the quality of the rendering depends on number of parameters (the XFree and the freetype versions and your video card(s)).

After an Xft font name you can add after a ";" an XLFD font name (simple or extended) as:

xft:Verdana:pixelsize=14;−adobe−courier−bold−r−normal−−14−*

then, if either loading the Xft font fails or fvwm has no Xft support, fvwm loads the font "−adobe−courier−bold−r−normal−−14−*". This allows for writing portable configuration files.

Font and string encoding

Once a font is loaded, fvwm finds its encoding (or charset) using its name (the last two fields of the name). fvwm assumes that the strings which are displayed with this font use this encoding (an exception is that if an iso10646−1 font is loaded, then UTF−8 is assumed for string encoding). In a normal situation, (i) a font is loaded by giving a font name without specifying the encoding, (ii) the encoding of the loaded font is the locale encoding, and then (iii) the strings in the fvwm configuration files should use the locale encoding as well as the window and icon name. With Xft the situation is bit different as Xft supports only iso10646−1 and iso8859−1. If you do not specify one of these encodings in the Xft font name, then fvwm does strings conversion using (iii). Note that with multibyte fonts (and in particular with "CJK" fonts) for good text rendering, the locale encoding should be the charset of the font.

To override the previous rules, it is possible to specify the string encoding in the beginning of a font description as follow:

StringEncoding=*enc*:_full_font_name_

where *enc* is an encoding supported by fvwm (usually font name charset plus some unicode encodings: UTF−8, USC−2, USC−4 and UTF−16).

For example, you may use an iso8859−1 locale charset and have an **FvwmForm** in Russian using koi8−r encoding. In this case, you just have to ask **FvwmForm** to load a koi8−r font by specifying the encoding in the font name. With a multibyte language, (as multibyte font works well only if the locale encoding is the charset of the font), you should use an iso10646−1 font:

StringEncoding=jisx0208.1983−0:−*−fixed−medium−r−*−ja−*−iso10646−1

or

"StringEncoding=jisx0208.1983−0:xft:Bitstream Cyberbit"

if your **FvwmForm** configuration uses jisx0208.1983−0 encoding. Another possibility is to use UTF−8 encoding for your **FvwmForm** configuration and use an iso10646−1 font:

−*−fixed−medium−r−*−ja−*−iso10646−1

or

"StringEncoding=UTF−8:xft:Bitstream Cyberbit"

or equivalently

"xft:Bitstream Cyberbit:encoding=iso10646−1"

In general iso10646−1 fonts together with UTF−8 string encoding allows the display of any characters in a given menu, **FvwmForm** etc.

More and more, unicode is used and text files use UTF−8 encoding. However, in practice the characters used range over your locale charset (this is the case when you generate a menu with fvwm−menu−desktop with recent versions of KDE and GNOME). For saving memory (an iso10646−1 font may have a very large number of characters) or because you have a pretty font without an iso10646−1 charset, you can specify the string encoding to be UTF−8 and use a font in the locale charset:

StringEncoding=UTF−8:−*−pretty_font−*−12−*

In most cases, fvwm correctly determines the encoding of the font. However, some fonts do not end with valid encoding names. When the font name isn't normal, for example:

−misc−fixed−*−−20−*−my_utf8−36

you need to add the encoding after the font name using a slash as a delimiter. For example:

MenuStyle * *Font* −misc−fixed−*−−20−*−my_utf8−36/iso10646−1

If fvwm finds an encoding, fvwm uses the iconv system functions to do conversion between encodings. Unfortunately, there are no standards. For conversion between iso8859−1 and UTF−8: a GNU system uses "ISO−8859−1" and other systems use "iso881" to define the converters (these two names are supported by fvwm). Moreover, in some cases it may be necessary to use machine specific converters. So, if you experience problems you can try to get information on your iconv implementation ("man iconv" may help) and put the name which defines the converter between the font encoding and UTF−8 at the end of the font name after the encoding hint and a / (another possible solution is to use GNU libiconv). For example use:

Style * *Font* −misc−fixed−*−−14−*−iso8859−1/*/latin1

to use latin1 for defining the converter for the iso8859−1 encoding. The "*" in between the "/" says to fvwm to determine the encoding from the end of the font name. Use:

Style * *Font* \
−misc−fixed−*−−14−*−local8859−6/iso8859−6/local_iso8859_6_iconv

to force fvwm to use the font with iso8859−6 as the encoding (this is useful for bi−directionality) and to use local_iso8859_6_iconv for defining the converters.

Font Shadow Effects

Fonts can be given 3d effects. At the beginning of the font name (or just after a possible StringEncoding specification) add

Shadow=*size* [*offset*] [*directions]*]:

size is a positive integer which specifies the number of pixels of shadow. *offset* is an optional positive integer which defines the number of pixels to offset the shadow from the edge of the character. The default offset is zero. *directions* is an optional set of directions the shadow emanates from the character. The *directions* are a space separated list of fvwm directions:

N, North, Top, t, Up, u, −

E, East, Right, r, Right, r,]

S, South, Bottom, b, Down, d, _

W, West, Left, l, Left, l, [

NE, NorthEast, TopRight, tr, UpRight, ur, ^

SE, SouthEast, BottomRight, br, DownRight, dr, >

SW, SouthWest, BottomLeft, bl, DownLeft, dl, v

NW, NorthWest, TopLeft, tl, UpLeft, ul, <

C, Center, Centre, .

A shadow is displayed in each given direction. *All* is equivalent to all the directions. The default *direction* is *BottomRight*. With the *Center* direction, the shadow surrounds the whole string. Since this is a super set of all other directions, it is a waste of time to specify this along with any other directions.

The shadow effect only works with colorsets. The color of the shadow is defined by using the *fgsh* option of the **Colorset** command. Please refer to the **Colorsets** section for details about colorsets.

Note: It can be difficult to find the font, *fg*, *fgsh* and *bg* colors to make this effect look good, but it can look quite good.

BI−DIRECTIONAL TEXT

Arabic and Hebrew text require bi−directional text support to be displayed correctly, this means that logical strings should be converted before their visual presentation, so left−to−right and right−to−left sub−strings are determined and reshuffled. In fvwm this is done automatically in window titles, menus, module labels and other places if the fonts used for displaying the text are of one of the charsets that require *bidi* (bi−directional) support. For example, this includes iso8859−6, iso8859−8 and iso10646−1 (unicode), but not other iso8859−* fonts.

This bi−directional text support is done using the *fribidi* library compile time option, see *INSTALL.fvwm*.

KEYBOARD SHORTCUTS

Almost all window manager operations can be performed from the keyboard so mouse−less operation should be possible. In addition to scrolling around the virtual desktop by binding the **Scroll** command to appropriate keys, **Popup**, **Move**, **Resize**, and any other command can be bound to keys. Once a command is started the pointer is moved by using the up, down, left, and right arrows, and the action is terminated by pressing return. Holding down the Shift key causes the pointer movement to go in larger steps and holding down the control key causes the pointer movement to go in smaller steps. Standard emacs and vi cursor movement controls (n , p , f , b , and j , k , h , l) can be used instead of the arrow keys.

SESSION MANAGEMENT

Fvwm supports session management according to the X Session Management Protocol. It saves and restores window position, size, stacking order, desk, stickiness, shadiness, maximizedness, iconifiedness for all windows. Furthermore, some global state is saved.

Fvwm doesn’t save any information regarding styles, decors, functions or menus. If you change any of these resources during a session (e.g. by issuing **Style** commands or by using various modules), these changes are lost after saving and restarting the session. To become permanent, such changes have to be added to the configuration file.

Note further that the current implementation has the following anomaly when used on a multi−screen display: Starting fvwm for the first time, fvwm manages all screens by forking a copy of itself for each screen. Every copy knows its parent and issuing a **Quit** command to any instance of fvwm kills the master and thus all copies of fvwm. When you save and restart the session, the session manager brings up a copy of fvwm on each screen, but this time they are started as individual instances managing one screen only. Thus a **Quit** kills only the copy it was sent to. This is probably not a very serious problem, since with session management, you are supposed to quit a session through the session manager anyway. If it is really needed,

Exec exec killall fvwm

still kills all copies of fvwm. Your system must have the **killall** command though.

BOOLEAN ARGUMENTS

A number of commands take one or several boolean arguments. These take a few equivalent inputs: "yes", "on", "true", "t" and "y" all evaluate to true while "no", "off", "false", "f" and "n" evaluate to false. Some commands allow "toggle" too which means that the feature is disabled if it is currently enabled and vice versa.

BUILTIN KEY AND MOUSE BINDINGS

The following commands are built−in to fvwm:

Key Help R A **Popup** MenuFvwmRoot
Key F1 R A **Popup** MenuFvwmRoot
Key Tab A M **WindowList** Root c c NoDeskSort
Key Escape A MC **EscapeFunc**
Mouse 1 R A **Menu** MenuFvwmRoot
Mouse 1 T A FuncFvwmRaiseLowerX **Move**

Mouse 1 FS A FuncFvwmRaiseLowerX **Resize**
Mouse 2 FST A FuncFvwmRaiseLowerX **Move**
AddToFunc FuncFvwmRaiseLowerX
+ I **Raise**
+ M $0
+ D **Lower**

The Help and F1 keys invoke a built−in menu that fvwm creates. This is primarily for new users that have not created their own configuration file. Either key on the root (background) window pops up an menu to help you get started.

The Tab key pressed anywhere with the Meta key (same as the Alt key on PC keyboards) held down pop−ups a window list.

Mouse button 1 on the title−bar or side frame can move, raise or lower a window.

Mouse button 1 on the window corners can resize, raise or lower a window.

You can override or remove these bindings. To remove the window list binding, use this:

Key Tab A M −

COMMAND EXECUTION

Module and Function Commands

If fvwm encounters a command that it doesn't recognize, it checks to see if the specified command should have been

Function (rest of command)

or

Module (rest of command)

This allows complex functions or modules to be invoked in a manner which is fairly transparent to the configuration file.

Example: the *config* file contains the line

HelpMe

Fvwm looks for an fvwm command called "HelpMe", and fails. Next it looks for a user−defined complex function called "HelpMe". If no such function exists, fvwm tries to execute a module called "HelpMe".

Delayed Execution of Commands

Note: There are many commands that affect look and feel of specific, some or all windows, like **Style**, **Mouse**, **Colorset**, **TitleStyle** and many others. For performance reasons such changes are not applied immediately but only when fvwm is idle, i.e. no user interaction or module input is pending. Specifically, new **Style** options that are set in a function are not applied until after the function has completed. This can sometimes lead to unwanted effects.

To force that all pending changes are applied immediately, use the **UpdateStyles**, **Refresh** or **RefreshWindow** commands.

QUOTING

Quotes are required only when needed to make fvwm consider two or more words to be a single argument. Unnecessary quoting is allowed. If you want a quote character in your text, you must escape it by using the backslash character. For example, if you have a pop−up menu called "Window−Ops", then you do not need quotes:

Popup Window−Ops

but if you replace the dash with a space, then you need quotes:

Popup "Window Ops"

The supported quoting characters are double quotes, single quotes and reverse single quotes. All three kinds of quotes are treated in the same way. Single characters can be quoted with a preceding backslash. Quoting single characters works even inside other kinds of quotes.

COMMAND EXPANSION

Whenever an fvwm command line is executed, fvwm performs parameter expansion. A parameter is a '$' followed by a word enclosed in brackets ($[...]) or a single special character. If fvwm encounters an unquoted parameter on the command line it expands it to a string indicated by the parameter name. Unknown parameters are left untouched. Parameter expansion is performed before quoting. To get a literal '$' use "$$".

If a command is prefixed with a '−' parameter expansion isn't performed. This applies to the command immediately following the '−', in which the expansion normally would have taken place. When uesed together with other prefix commands it must be added before the other prefix.

Example:

Pick −**Exec** exec xmessage '$[w.name]'

opens an xmessage dialog with "$[w.name]" unexpanded.

The longer variables may contain additional variables inside the name, which are expanded before the outer variable.

In earlier versions of fvwm, some single letter variables were supported. It is deprecated now, since they cause a number of problems. You should use the longer substitutes instead.

Example:

```
# Print the current desk number, horizontal page number
# and the window's class (unexpanded here, no window).
Echo $[desk.n] $[page.nx] $[w.class]
```

Note: If the command is called outside a window context, it prints "$[w.class]" instead of the class name. It is usually not enough to have the pointer over a window to have a context window. To force using the window with the focus, the **Current** command can be used:

Current Echo $[desk.n] $[page.nx] $[w.class]

The parameters known by fvwm are:

$$
A literal '$'.

$.
The absolute directory of the currently Read file. Intended for creating relative and relocatable configuration trees. If used outside of any read file, the returned value is '.'.

$0 to $9
The positional parameters given to a complex function (a function that has been defined with the **AddToFunc** command). "$0" is replaced with the first parameter, "$1" with the second parameter and so on. If the corresponding parameter is undefined, the "$..." is deleted from the command line.

$*
All positional parameters given to a complex function. This includes parameters that follow after "$9".

$[*n*]
The *n*:th positional parameter given to a complex function, counting from 0. If the corresponding parameter is undefined, the "$[*n*]" is deleted from the command line. The parameter is expanded unquoted.

$[*n*−*m*]
The positional parameters given to a complex function, starting with parameter *n* and ending with parameter *m*. If all the corresponding parameters are undefined, the "$[...]" is deleted from the command line. If only some of the parameters are defined, all defined parameters are expanded, and the remaining silently ignored. All parameters are expanded unquoted.

$[*n*−]
All the positional parameters given to a complex function, starting with parameter *n*. If all the corresponding parameters are undefined, the "$[...]" is deleted from the command line. All parameters are expanded unquoted.

$[*]
All the positional parameters given to a complex function. This is equivalent of $[0−].

$[version.num]
The version number, like "2.6.0".

$[version.info]
The version info, like " (from cvs)", empty for the official releases.

$[version.line]
The first line printed by the −−version command line option.

$[vp.x] $[vp.y] $[vp.width] $[vp.height]
Either coordinate or the width or height of the current viewport.

$[wa.x] $[wa.y] $[wa.width] $[wa.height]
Either coordinate or the width or height of the EWMH working area.

$[dwa.x] $[dwa.y] $[dwa.width] $[dwa.height]
Either coordinate or the width or height of the dynamic EWMH working area.

$[desk.n]
The current desk number.

$[desk.name<n>]
These parameters are replaced with the name of the desktop number <n> that is defined with the **DesktopName** command. If no name is defined, then the default name is returned.

$[desk.width] $[desk.height]
The width or height of the whole desktop, i.e. the width or height multiplied by the number of pages in x or y direction.

$[desk.pagesx] $[desk.pagesy]
The number of total pages in a desk in x or y direction. This is the same as the values set by **DesktopSize**.

$[page.nx] $[page.ny]
The current page numbers, by X and Y axes, starting from 0. *page* is equivalent to *area* in the GNOME terminology.

$[w.id]
The window−id (expressed in hex, e.g. 0x10023c) of the window the command was called for or "$[w.id]" if no window is associated with the command.

$[w.name] $[w.iconname] $[w.class] $[w.resource] $[w.visiblename] $[w.iconfile] $[w.miniiconfile] $[w.iconfile.svgopts] $[w.miniiconfile.svgopts]
The window's name, icon name, resource class and resource name, visible name, file name of its icon or mini icon defined with the *Icon* or *MiniIcon* style (including the full path if the file was found on disk), and (if fvwm is compiled with SVG support) the icon or mini icon svg rendering options (including the leading colon), or unexpanded "$[w.<attribute>]" string if no window is associated with the command.

Note, the first 5 variables may include any kind of characters, so these variables are quoted. It means that the value is surrounded by single quote characters and any contained single quote is prefixed with a backslash. This guarantees that commands like:

Style $[w.resource] *Icon* norm/network.png

work correctly, regardless of any special symbols the value may contain, like spaces and different kinds of quotes.

In the case of the window's visible name, this is the value returned from the literal title of the window shown in the titlebar. Typically this will be the same as $[w.name] once expanded, although in the case of using *IndexedWindowName* then this is more useful a distinction, and allows for referencing the specific window by its visible name for inclusion in things like **Style** commands.

$[w.x] $[w.y] $[w.width] $[w.height]
Either coordinate or the width or height of the current window if it is not iconified. If no window is associated with the command or the window is iconified, the string is left as is.

$[w.desk]
The number of the desk on which the window is shown. If the window is sticky the current desk number is used.

$[w.layer]
The layer of the window.

$[w.screen]
The screen number the window is on. If Xinerama is not present, this returns the number 0.

$[cw.x] $[cw.y] $[cw.width] $[cw.height]
These work like $[w....] but return the geometry of the client part of the window. In other words: the border and title of the window is not taken into account.

$[i.x], $[it.x], $[ip.x] $[i.y], $[it.y], $[ip.y] $[i.width], $[it.width], $[ip.width] $[i.height], $[it.height], $[ip.height]
These work like $[w....] but return the geometry of the icon ($[i....]), the icon title ($[it....]) or the icon picture ($[ip....]).

$[pointer.x] $[pointer.y]
These return the position of the pointer on the screen. If the pointer is not on the screen, these variables are not expanded.

$[pointer.wx] $[pointer.wy]
These return the position of the pointer in the selected window. If the pointer is not on the screen, the window is iconified or no window is selected, these variables are not expanded.

$[pointer.cx] $[pointer.cy]
These return the position of the pointer in the client portion of the selected window. If the pointer is not on the screen, the window is shaded or iconified or no window is selected, these variables are not expanded.

$[pointer.screen]
The screen number the pointer is currently on. Returns 0 if Xinerama is not enabled.

$[screen]
The screen number fvwm is running on. Useful for setups with multiple screens.

$[fg.cs<n>] $[bg.cs<n>] $[hilight.cs<n>] $[shadow.cs<n>]
These parameters are replaced with the name of the foreground (fg), background (bg), hilight (hilight) or shadow (shadow) color that is defined in colorset <n> (replace <n> with zero or a positive integer). For example "$[fg.cs3]" is expanded to the name of the foreground color of colorset 3 (in rgb:rrrr/gggg/bbbb form). Please refer to the **Colorsets** section for details about colorsets.

$[schedule.last]
This is replaced by the id of the last command that was scheduled with the **Schedule** command, even if this command was already executed.

$[schedule.next]
This is replaced by the id the next command used with **Schedule** will get (unless a different id is specified explicitly).

$[cond.rc]
The return code of the last conditional command. This variable is only valid inside a function and can not be used in a conditional command. Please refer to the section **Conditional Commands** in the command list.

$[func.context]
The context character of the running command as used in the **Mouse**, **Key** or **PointerKey** command. This is useful for example with:

Mouse 3 FS N **WindowShade** $$[func.context]

$[gt.*str*]
return the translation of *str* by looking in the current locale catalogs. If no translation is found *str* is returned as is. See the **LocalePath** command.

$[infostore.*key*]
Return the value of the item stored in the InfoStore at the given *key*. If no key is present, the unexpanded string is returned.

$[...]
If the string within the braces is neither of the above, fvwm tries to find an environment variable with this name and replaces its value if one is found (e.g. "$[PAGER]" could be replaced by "more"). Otherwise the string is left as is.

Some examples can be found in the description of the **AddToFunc** command.

SCRIPTING & COMPLEX FUNCTIONS

To achieve the more complex effects, fvwm has a number of commands that improve its scripting abilities. Scripts can be read from a file with **Read**, from the output of a command with **PipeRead** or written as a complex function with the **AddToFunc** command. For the curious, section 7 of the fvwm FAQ shows some real life applications of scripting. Please refer to the sections **User Functions and Shell Commands** and **Conditional Commands** for details. A word of warning: during execution of complex functions, fvwm needs to take all input from the mouse pointer (the pointer is "grabbed" in the slang of X). No other programs can receive any input from the pointer while a function is run. This can confuse some programs. For example, the xwd program refuses to make screen shots when run from a complex function. To achieve the same functionality you can use the **Read** or **PipeRead** command instead.

LIST OF FVWM COMMANDS

The command descriptions below are grouped together in the following sections. The sections are hopefully sorted in order of usefulness to the newcomer.

- **Menu commands**
- **Miscellaneous commands**
- **Commands affecting window movement and placement**
- **Commands for focus and mouse movement**

- **Commands controlling window state**
- **Commands for mouse, key and stroke bindings**
- **The Style command (controlling window styles)**
- **Other commands controlling window styles**
- **Commands controlling the virtual desktop**
- **Commands for user functions and shell commands**
- **Conditional commands**
- **Module commands**
- **Quit, restart and session management commands**
- **Colorsets**
- **Color gradients**

Menus

Before a menu can be opened, it has to be populated with menu items using the **AddToMenu** command and bound to a key or mouse button with the **Key**, **PointerKey** or **Mouse** command (there are many other ways to invoke a menu too). This is usually done in the configuration file.

Fvwm menus are extremely configurable in look and feel. Even the slightest nuances can be changed to the user's liking, including the menu item fonts, the background, delays before popping up sub menus, generating menus dynamically and many other features. Please refer to the **MenuStyle** command to learn more.

Types of Menus

In fvwm there are four slightly different types of menus:

Popup menus can appear everywhere on the screen on their own or attached to a part of a window. The **Popup** command opens popup menus. If the popup menu was invoked with a mouse button held down, it is closed when the button is released. The item under the pointer is then activated and the associated action is executed.

Menu is a very similar command, but the menus it opens are slightly less transient. When invoked by clicking a mouse button, it stays open and can be navigated with no button held. But if it is invoked by a button press followed by mouse motion, it behaves exactly like a popup menu.

Tear off menus or *Pin up menus* are menus from either of the above two commands that have been "torn off" their original context and pinned on the desktop like a normal window. They are created from other menus by certain key presses or mouse sequences or with the **TearMenuOff** command

from inside a menu.

Sub menus are menus inside menus. When a menu item that has the **Popup** command as its action is selected, the named menu is opened as an inferior menu to the parent. Any type of menu can have sub menus.

Menu Anatomy

Menus consist of any number of titles which are inactive menu items that usually appear at the top of the menu, normal items triggering various actions when selected, separator lines between the items, tear off bars (a horizontal broken line) that tear off the menu when selected, and sub menu items indicated with a triangle pointing left or right, depending on the direction in which the sub menu appears. All the above menu items are optional.

Additionally, if the menu is too long to fit on the screen, the excess menu items are put in a continuation menu and a sub menu with the string "More..." is placed at the bottom of the menu. The "More..." string honors the locale settings.

Finally, there may be a picture running up either side of the menu (a "side bar").

Menu Navigation

Menus can be navigated either with the keyboard or with the mouse. Many people prefer to use the mouse, but it can be rather tedious. Once you get the hang of it, keyboard navigation can be much faster. While fvwm displays a menu, it can do nothing else. For example, new windows do not appear before the menu is closed. However, this is not exactly true for tear off menus. See the **Tear Off Menus** section for details.

Mouse Navigation

Moving the pointer over a menu selects the item below it. Normally this is indicated by a 3d border around the item, but not all parts of a menu can be selected. Pressing any mouse button while a menu is open by default activates the item below it. Items of a popup menu are also activated by releasing a held mouse button. In case of an item that hides a sub menu, the sub menu is displayed if the pointer hovers over the item long enough or moves close to the triangle indicating the sub menu. This behaviour can be tuned with menu styles.

Scrolling a mouse wheel over a menu either wraps the pointer along the menu (default), scrolls the menu under the pointer or act as if the menu was clicked depending on the *MouseWheel* menu style.

Clicking on a selected item activates it − what happens exactly depends on the type of the item.

Clicking on a title, a separator, the side bar, or outside the menu closes the menu (exception: tear off menus can not be closed this way). Pressing mouse button 2 over a menu title or activating a tear off bar creates a tear off menu from the current menu. Clicking on a normal menu item invokes the command that is bound to it, and clicking on a sub menu item either closes all open menus and replaces them with the sub menu or posts the menu (default).

Posting menus is meant to ease mouse navigation. Once a sub menu is posted, only items from that sub menu can be selected. This can be very useful to navigate the menu if the pointer tends to stray off the menu. To unpost the menu and revert back to normal operation, either click on the same sub menu item or press any key.

Keyboard Navigation

Just like with mouse navigation, the item below the pointer is selected. This is achieved by warping the pointer to the menu items when necessary. While a menu is open, all key presses are intercepted by the menu. No other application can get keyboard input (although this is not the case for tear off menus).

Items can be selected directly by pressing a hotkey that can be configured individually for each menu item. The hotkey is indicated by underlining it in the menu item label. With the *AutomaticHotkeys* menu style fvwm automatically assigns hotkeys to all menu items.

The most basic keys to navigate through menus are the cursor keys (move up or down one item,

enter or leave a sub menu), Space (activate item) and Escape (close menu). Numerous other keys can be used to navigate through menus by default:

Enter, *Return*, *Space* activate the current item.

Escape, *Delete*, *Ctrl−G* exit the current sequence of menus or destroy a tear off menu.

J, *N*, *Cursor−Down*, *Tab*, *Meta−Tab*, *Ctrl−F*, move to the next item.

K, *P*, *Cursor−Up*, *Shift−Tab*, *Shift−Meta−Tab*, *Ctrl−B*, move to the prior item.

L, *Cursor−Right*, *F* enter a sub menu.

H, *Cursor−Left*, *B* return to the prior menu.

Ctrl−Cursor−Up, *Ctrl−K Ctrl−P*, *Shift−Ctrl−Meta−Tab*, *Page−Up* move up five items.

Ctrl−Cursor−Down, *Ctrl−J Ctrl−N*, *Ctrl−Meta−Tab Page−Down* move down five items.

Shift−P, *Home*, *Shift−Cursor−Up*, *Ctrl−A* move to the first item.

Shift−N, *End*, *Shift−Cursor−Down*, *Ctrl−E* move to the last item.

Meta−P, *Meta−Cursor−Up*, *Ctrl−Cursor−Left*, *Shift−Ctrl−Tab*, move up just below the next separator.

Meta−N, *Meta−Cursor−Down*, *Ctrl−Cursor−Right*, *Ctrl−Tab*, move down just below the next separator.

Insert opens the "More..." sub menu if any.

Backspace tears off the menu.

Menu Bindings

The keys and mouse buttons used to navigate the menu can be configured using the **Key** and **Mouse** commands with the special context ’M’, possible combined with ’T’ for the menu title, ’I’ for other menu items, ’S’ for any border or sidepic, ’[’ for left border including a left sidepic, ’]’ for right border including a right sidepic, ’−’ for top border, ’_’ for bottom border. The menu context uses its own set of actions that can be bound to keys and mouse buttons. These are *MenuClose*, *MenuCloseAndExec*, *MenuEnterContinuation*, *MenuEnterSubmenu*, *MenuLeaveSubmenu*, *MenuMoveCursor*, *MenuCursorLeft*, *MenuCursorRight*, *MenuSelectItem*, *MenuScroll* and *MenuTearOff*.

It is not possible to override the key Escape with no modifiers for closing the menu. Neither is it possible to undefine mouse button 1, the arrow keys or the enter key for minimal navigation.

MenuClose exits from the current sequence of menus or destroys a tear off menu.

MenuCloseAndExec exits from the current sequence of menus or destroys a tear off menu and executes the rest of the line as a command.

MenuEnterContinuation opens the "More..." sub menu if any.

MenuEnterSubmenu enters a sub menu.

MenuLeaveSubmenu returns to the prior menu.

MenuMoveCursor *n* [*m*] moves the selection to another item. If the first argument is zero the second argument specifies an absolute item in the menu to move the pointer to. Negative items are counted from the end of the menu. If the first argument is non−zero, the second argument must be omitted, and the first argument specifies a relative change in the selected item. The positions may be suffixed with a ’s’ to indicate that the items should refer only to the first items after separators.

MenuCursorLeft enters a sub menu with the *SubmenusLeft* menu style, and returns to the prior menu with the *SubmenusRight* menu style.

MenuCursorRight enters a sub menu with the *SubmenusRight* menu style, and returns to the prior menu with the *SubmenusLeft* menu style.

MenuSelectItem triggers the action for the menu item.

MenuScroll *n* performs menu scrolling according to the *MouseWheel* menu style with *n* items. The distance can be suffixed with an 's' to indicate the items should refer only to the first items after separators.

MenuTearOff turns a normal menu into a "torn off" menu. See **Tear Off Menus** for details.

Tear Off Menus

A tear off menu is any menu that has been "torn off" the window it was attached to and pinned to the root window. There are three ways to tear off a menu: click on the menu title with mouse button 2, press Backspace in the menu or activate its tear off bar (a horizontal bar with a broken line). Tear off bars must be added to the menu as any other item by assigning them the command **TearMenuOff**.

The builtin tear off actions can be overridden by undefining the builtin menu actions bound to tear off. To remove the builtin mouse button 2 binding, use:

Mouse 2 MT A −

and to remove the builtin backspace binding, use:

Key Backspace M A −

See the section **Menu Bindings** for details on how to assign other bindings for tear off.

Note that prior to fvwm 2.5.20 the tear off mouse bindings were redefined in different way, which no longer work.

The window containing the menu is placed as any other window would be. If you find it confusing to have your tear off menus appear at random positions on the screen, put this line in your configuration file:

Style fvwm_menu *UsePPosition*

To remove borders and buttons from a tear−off menu but keep the menu title, you can use

Style fvwm_menu !*Button* 0, !*Button* 1
Style fvwm_menu !*Button* 2, !*Button* 3
Style fvwm_menu !*Button* 4, !*Button* 5
Style fvwm_menu !*Button* 6, !*Button* 7
Style fvwm_menu !*Button* 8, !*Button* 9
Style fvwm_menu *Title*, *HandleWidth* 0

A tear off menu is a cross breeding between a window and a menu. The menu is swallowed by a window and its title is stripped off and displayed in the window title. The main advantage is that the menu becomes permanent − activating an item does not close the menu. Therefore, it can be used multiple times without reopening it. To destroy such a menu, close its window or press the Escape key.

Tear off menus behave somewhat differently than normal menus and windows. They do not take the keyboard focus, but while the pointer is over one of them, all key presses are sent to the menu. Other fvwm key bindings are disabled as long as the pointer is inside the tear off menu or one of its sub menus. When the pointer leaves this area, all sub menus are closed immediately. Note that the window containing a tear off menu is never hilighted as if it had the focus.

A tear off menu is an independent copy of the menu it originated from. As such, it is not affected by adding items to that menu or changing its menu style.

To create a tear off menu without opening the normal menu first, the option *TearOffImmediately* can be added to the **Menu** or **Popup** command.

AddToMenu *menu−name* [*menu−label action*]
Begins or adds to a menu definition. Typically a menu definition looks like this:

```
AddToMenu Utilities Utilities Title
+ Xterm          Exec  exec xterm −e tcsh
+ Rxvt           Exec  exec rxvt
+ "Remote Logins" Popup Remote−Logins
+ Top            Exec  exec rxvt −T Top −n Top −e top
+ Calculator     Exec  exec xcalc
+ Xman           Exec  exec xman
+ Xmag           Exec  exec xmag
+ emacs          Exec  exec xemacs
+ Mail           MailFunction xmh "−font fixed"
+ ""             Nop
+ Modules        Popup Module−Popup
+ ""             Nop
+ Exit Fvwm      Popup Quit−Verify
```

The menu could be invoked via

```
Mouse 1 R A Menu Utilities Nop
```

or

```
Mouse 1 R A Popup Utilities
```

There is no end−of−menu symbol. Menus do not have to be defined in a contiguous region of the *config* file. The quoted (or first word) portion in the above examples is the menu label, which appears in the menu when the user pops it up. The remaining portion is an fvwm command which is executed if the user selects that menu item. An empty menu−label ("") and the **Nop** function are used to insert a separator into the menu.

The keywords *DynamicPopUpAction* and *DynamicPopDownAction* have a special meaning when used as the name of a menu item. The action following the keyword is executed whenever the menu is popped up or down. This way you can implement dynamic menus. It is even possible to destroy itself with **DestroyMenu** and the rebuild from scratch. When the menu has been destroyed (unless you used the *recreate* option when destroying the menu), do not forget to add the dynamic action again.

Note: Do not trigger actions that require user interaction. They may fail and may screw up your menus. See the**Silent** command.

Warning
Do not issue **MenuStyle** commands as dynamic menu actions. Chances are good that this crashes fvwm.

There are several configurable scripts installed together with fvwm for automatic menu generation. They have their own man pages. Some of them, specifically **fvwm−menu−directory** and **fvwm−menu−desktop**, may be used with *DynamicPopupAction* to create a directory listing or GNOME/KDE application listing.

Example (File browser):

```
# You can find the shell script fvwm_make_browse_menu.sh
# in the utils/ directory of the distribution.
AddToMenu BrowseMenu
+ DynamicPopupAction PipeRead \
  'fvwm_make_browse_menu.sh BrowseMenu'
```

Example (Picture menu):

```
# Build a menu of all .jpg files in
# $HOME/Pictures
AddToMenu JpgMenu foo title
+ DynamicPopupAction Function MakeJpgMenu

AddToFunc MakeJpgMenu
+ I DestroyMenu recreate JpgMenu
+ I AddToMenu JpgMenu Pictures Title
+ I PipeRead 'for i in $HOME/Pictures/*.jpg; \
  do echo AddToMenu JpgMenu "`basename $i`" Exec xv $i; done'
```

The keyword *MissingSubmenuFunction* has a similar meaning. It is executed whenever you try to pop up a sub menu that does not exist. With this function you can define and destroy menus on the fly. You can use any command after the keyword, but if the name of an item (that is a submenu) defined with **AddToFunc** follows it, fvwm executes this command:

Function <function−name> <submenu−name>

i.e. the name is passed to the function as its first argument and can be referred to with "$0".

The **fvwm−menu−directory** script mentioned above may be used with *MissingSubmenuFunction* to create an up to date recursive directory listing.

Example:

```
# There is another shell script fvwm_make_directory_menu.sh
# in the utils/ directory of the distribution. To use it,
# define this function in your configuration file:

DestroyFunc MakeMissingDirectoryMenu
AddToFunc MakeMissingDirectoryMenu
+ I PipeRead fvwm_make_directory_menu.sh $0

DestroyMenu SomeMenu
AddToMenu SomeMenu
+ MissingSubmenuFunction MakeMissingDirectoryMenu
+ "Root directory" Popup /
```

This is another implementation of the file browser that uses sub menus for subdirectories.

Titles can be used within the menu. If you add the option *top* behind the keyword **Title**, the title is added to the top of the menu. If there was a title already, it is overwritten.

```
AddToMenu Utilities Tools Title top
```

All text up to the first Tab in the menu label is aligned to the left side of the menu, all text right of the first Tab is aligned to the left in a second column and all text thereafter is placed right aligned in the third column. All other Tab s are replaced by spaces. Note that you can change this format with the *ItemFormat* option of the **MenuStyle** command.

If the menu−label contains an ampersand ('&'), the next character is taken as a hot−key for the menu item. Hot−keys are underlined in the label. To get a literal '&', insert "&&". Pressing the hot−key moves through the list of menu items with this hot−key or selects an item that is the only one with this hot−key.

If the menu−label contains a sub−string which is set off by stars, then the text between the stars is expected to be the name of an image file to insert in the menu. To get a literal '*', insert "**". For example

+ Calculator*xcalc.xpm* **Exec** exec xcalc

inserts a menu item labeled "Calculator" with a picture of a calculator above it. The following:

+ *xcalc.xpm* **Exec** exec xcalc

Omits the "Calculator" label, but leaves the picture.

If the menu−label contains a sub−string which is set off by percent signs, then the text between the percent signs is expected to be the name of image file (a so called mini icon to insert to the left of the menu label. A second mini icon that is drawn at the right side of the menu can be given in the same way. To get a literal '%', insert "%%". For example

+ Calculator%xcalc.xpm% **Exec** exec xcalc

inserts a menu item labeled "Calculator" with a picture of a calculator to the left. The following:

+ %xcalc.xpm% **Exec** exec xcalc

Omits the "Calculator" label, but leaves the picture. The pictures used with this feature should be small (perhaps 16x16).

If the menu−name (not the label) contains a sub−string which is set off by at signs ('@'), then the text between them is expected to be the name of an image file to draw along the left side of the menu (a side pixmap). You may want to use the *SidePic* option of the **MenuStyle** command instead. To get a literal '@', insert "@@". For example

AddToMenu StartMenu@linux−menu.xpm@

creates a menu with a picture in its bottom left corner.

If the menu−name also contains a sub−string surrounded by '^'s, then the text between '^'s is expected to be the name of an X11 color and the column containing the side picture is colored with that color. You can set this color for a menu style using the *SideColor* option of the **MenuStyle** command. To get a literal '^', insert "^^". Example:

AddToMenu StartMenu@linux−menu.xpm@^blue^

creates a menu with a picture in its bottom left corner and colors with blue the region of the menu containing the picture.

In all the above cases, the name of the resulting menu is name specified, stripped of the substrings between the various delimiters.

ChangeMenuStyle *menustyle menu ...*
Changes the menu style of *menu* to *menustyle*. You may specify more than one menu in each call of **ChangeMenuStyle**.

CopyMenuStyle *orig−menustyle dest−menustyle*
Copy *orig−menustyle* to *dest−menustyle*, where *orig−menustyle* is an existing menu style. If the menu style *dest_menustyle* does not exist, then it is created.

DestroyMenu [recreate] *menu*
Deletes a menu, so that subsequent references to it are no longer valid. You can use this to change the contents of a menu during an fvwm session. The menu can be rebuilt using **AddToMenu**. The optional parameter *recreate* tells fvwm not to throw away the menu completely but to throw away all the menu items (including the title).

DestroyMenu Utilities

DestroyMenuStyle *menustyle*
Deletes the menu style named *menustyle* and changes all menus using this style to the default style, you cannot destroy the default menu style.

DestroyMenuStyle pixmap1

Menu *menu−name* [*position*] [*double−click−action*]

Causes a previously defined menu to be popped up in a sticky manner. That is, if the user invokes the menu with a click action instead of a drag action, the menu stays up. The command *double−click−action* is invoked if the user double−clicks a button (or hits the key rapidly twice if the menu is bound to a key) when bringing up the menu. If the double click action is not specified, double clicking on the menu does nothing. However, if the menu begins with a menu item (i.e. not with a title or a separator) and the double click action is not given, double clicking invokes the first item of the menu (but only if the pointer really was over the item).

The pointer is warped to where it was when the menu was invoked if it was both invoked and closed with a keystroke.

The *position* arguments allow placement of the menu somewhere on the screen, for example centered on the visible screen or above a title bar. Basically it works like this: you specify a *context−rectangle* and an offset to this rectangle by which the upper left corner of the menu is moved from the upper left corner of the rectangle. The *position* arguments consist of several parts:

[*context−rectangle*] *x y* [*special−options*]

The *context−rectangle* can be one of:

Root
the root window of the current screen.

XineramaRoot
the root window of the whole Xinerama screen. Equivalent to "root" when Xinerama is not used.

Mouse
a 1x1 rectangle at the mouse position.

Window
the frame of the context window.

Interior
the inside of the context window.

Title
the title of the context window or icon.

Button<n>
button #n of the context window.

Icon
the icon of the context window.

Menu
the current menu.

Item
the current menu item.

Context
the current window, menu or icon.

This
whatever widget the pointer is on (e.g. a corner of a window or the root window).

Rectangle <geometry>
the rectangle defined by *<geometry>* in X geometry format. Width and height default to 1 if omitted.

If the context−rectangle is omitted or illegal (e.g. "item" on a window), "Mouse" is the default. Note that not all of these make sense under all circumstances (e.g. "Icon" if the pointer is on a

menu).

The offset values *x* and *y* specify how far the menu is moved from its default position. By default, the numeric value given is interpreted as a percentage of the context rectangle's width (height), but with a trailing '*m*' the menu's width (height) is used instead. Furthermore a trailing '*p*' changes the interpretation to mean pixels.

Instead of a single value you can use a list of values. All additional numbers after the first one are separated from their predecessor by their sign. Do not use any other separators.

If *x* or *y* are prefixed with "'*o*<number>" where <number> is an integer, the menu and the rectangle are moved to overlap at the specified position before any other offsets are applied. The menu and the rectangle are placed so that the pixel at <number> percent of the rectangle's width/height is right over the pixel at <number> percent of the menu's width/height. So "o0" means that the top/left borders of the menu and the rectangle overlap, with "o100" it's the bottom/right borders and if you use "o50" they are centered upon each other (try it and you will see it is much simpler than this description). The default is "o0". The prefix "o<number>" is an abbreviation for "+<number>−<number>m".

A prefix of '*c*' is equivalent to "o50". Examples:

```
# window list in the middle of the screen
WindowList Root c c

# menu to the left of a window
Menu name window −100m c+0

# popup menu 8 pixels above the mouse pointer
Popup name mouse c −100m−8p

# somewhere on the screen
Menu name rectangle 512x384+1+1 +0 +0

# centered vertically around a menu item
AddToMenu foobar−menu
 + "first item" Nop
 + "special item" Popup "another menu" item +100 c
 + "last item" Nop

# above the first menu item
AddToMenu foobar−menu
 + "first item" Popup "another menu" item +0 −100m
```

Note that you can put a sub menu far off the current menu so you could not reach it with the mouse without leaving the menu. If the pointer leaves the current menu in the general direction of the sub menu the menu stays up.

The *special−options*:

To create a tear off menu without opening the normal menu, add the option *TearOffImmediately*. Normally the menu opens in normal state for a split second before being torn off. As tearing off places the menu like any other window, a position should be specified explicitly:

```
# Forbid fvwm to place the menu window
Style <name of menu> UsePPosition
# Menu at top left corner of screen
Menu Root 0p 0p TearOffImmediately
```

The *Animated* and *Mwm* or *Win* menu styles may move a menu somewhere else on the screen. If

you do not want this you can add *Fixed* as an option. This might happen for example if you want the menu always in the top right corner of the screen.

Where do you want a menu to appear when you click on its menu item? The default is to place the title under the cursor, but if you want it where the position arguments say, use the *SelectInPlace* option. If you want the pointer on the title of the menu, use *SelectWarp* too. Note that these options apply only if the *PopupAsRootMenu* **MenuStyle** option is used.

The pointer is warped to the title of a sub menu whenever the pointer would be on an item when the sub menu is popped up (*fvwm* menu style) or never warped to the title at all (*Mwm* or *Win* menu styles). You can force (forbid) warping whenever the sub menu is opened with the *WarpTitle* (*NoWarp*) option.

Note that the *special−options* do work with a normal menu that has no other position arguments.

MenuStyle *stylename* [*options*]

Sets a new menu style or changes a previously defined style. The *stylename* is the style name; if it contains spaces or tabs it has to be quoted. The name "*" is reserved for the default menu style. The default menu style is used for every menu−like object (e.g. the window created by the **WindowList** command) that had not be assigned a style using the **ChangeMenuStyle**. See also **DestroyMenuStyle**. When using monochrome color options are ignored.

options is a comma separated list containing some of the keywords Fvwm / Mwm / Win, BorderWidth, Foreground, Background, Greyed, HilightBack / !HilightBack, HilightTitleBack, ActiveFore / !ActiveFore, MenuColorset, ActiveColorset, GreyedColorset, TitleColorset, Hilight3DThick / Hilight3DThin / Hilight3DOff, Hilight3DThickness, Animation / !Animation, Font, TitleFont, MenuFace, PopupDelay, PopupOffset, TitleWarp / !TitleWarp, TitleUnderlines0 / TitleUnderlines1 / TitleUnderlines2, SeparatorsLong / SeparatorsShort, TrianglesSolid / TrianglesRelief, PopupImmediately / PopupDelayed, PopdownImmediately / PopdownDelayed, PopupActiveArea, DoubleClickTime, SidePic, SideColor, PopupAsRootMenu / PopupAsSubmenu / PopupIgnore / PopupClose, RemoveSubmenus / HoldSubmenus, SubmenusRight / SubmenusLeft, SelectOnRelease, ItemFormat, VerticalItemSpacing, VerticalMargins, VerticalTitleSpacing, AutomaticHotkeys / !AutomaticHotkeys, UniqueHotkeyActivatesImmediate / !UniqueHotkeyActivatesImmediate, MouseWheel, ScrollOffPage / !ScrollOffPage, TrianglesUseFore / !TrianglesUseFore.

In the above list some options are listed as option pairs or triples with a '/' in between. These options exclude each other. All paired options can be negated to have the effect of the counterpart option by prefixing ! to the option.

Some options are now negated by prefixing ! to the option. This is the preferred form for all such options. The other negative forms are now deprecated and will be removed in the future.

This is a list of MenuStyle deprecated negative options: ActiveForeOff, AnimationOff, AutomaticHotkeysOff, HilightBackOff, TitleWarpOff

Fvwm, *Mwm*, *Win* reset all options to the style with the same name in former versions of fvwm. The default for new menu styles is *Fvwm* style. These options override all others except *Foreground*, *Background*, *Greyed*, *HilightBack*, *ActiveFore* and *PopupDelay*, so they should be used only as the first option specified for a menu style or to reset the style to defined behavior. The same effect can be created by setting all the other options one by one.

Mwm and *Win* style menus popup sub menus automatically. *Win* menus indicate the current menu item by changing the background to dark. *Fvwm* sub menus overlap the parent menu, *Mwm* and *Win* style menus never overlap the parent menu.

Fvwm style is equivalent to !HilightBack, Hilight3DThin, !ActiveFore, !Animation, Font, MenuFace, PopupOffset 0 67, TitleWarp, TitleUnderlines1, SeparatorsShort, TrianglesRelief, PopupDelayed, PopdownDelayed, PopupDelay 150, PopdownDelay 150, PopupAsSubmenu, HoldSubmenus, SubmenusRight, BorderWidth 2, !AutomaticHotkeys, UniqueHotkeyActivatesImmediate, PopupActiveArea 75.

Mwm style is equivalent to !HilightBack, Hilight3DThick, !ActiveFore, !Animation, Font, MenuFace, PopupOffset −3 100, !TitleWarp, TitleUnderlines2, SeparatorsLong, TrianglesRelief, PopupImmediately, PopdownDelayed, PopdownDelay 150, PopupAsSubmenu, HoldSubmenus, SubmenusRight, BorderWidth 2, UniqueHotkeyActivatesImmediate, !AutomaticHotkeys, PopupActiveArea 75.

Win style is equivalent to HilightBack, Hilight3DOff, ActiveFore, !Animation, Font, MenuFace, PopupOffset −5 100, !TitleWarp, TitleUnderlines1, SeparatorsShort, TrianglesSolid, PopupImmediately, PopdownDelayed, PopdownDelay 150, PopupAsSubmenu, RemoveSubmenus, SubmenusRight, BorderWidth 2, UniqueHotkeyActivatesImmediate, !AutomaticHotkeys, PopupActiveArea 75.

BorderWidth takes the thickness of the border around the menus in pixels. It may be zero to 50 pixels. The default is 2. Using an illegal value reverts the border width to the default.

Foreground and *Background* may have a color name as an argument. This color is used for menu text or the menu's background. You can omit the color name to reset these colors to the built−in default.

Greyed may have a color name as an argument. This color is the one used to draw a menu−selection which is prohibited (or not recommended) by the Mwm hints which an application has specified. If the color is omitted the color of greyed menu entries is based on the background color of the menu.

HilightBack and *!HilightBack* switch hilighting the background of the selected menu item on and off. A specific background color may be used by providing the color name as an argument to *HilightBack*. If you use this option without an argument the color is based on the menu's background color. The*ActiveColor set* option overrides the specified color. If the colorset has a non solid background it is used for the hilighting.

HilightTitleBack switches hilighting the background of menu titles on. If a *TitleColorset* was used, the background colour is taken from there. Otherwise the color is based on the menu's background color. If the colorset has a non solid background it is used for the hilighting.

ActiveFore and *!ActiveFore* switch hilighting the foreground of the selected menu item on and off. A specific foreground color may be used by providing the color name as an argument to *ActiveFore*. Omitting the color turns hilighting on when an*ActiveColor set* is used. *ActiveFore* turns off hilighting the foreground completely. The*ActiveColor set* option overrides the specified color.

MenuColorset controls if a colorset is used instead of the *Foreground*, *Background* and *MenuFace* menu styles. If the *MenuColorset* keyword is followed by a number equal to zero or greater, this number is taken as the number of the colorset to use. If the number is omitted, the colorset is switched off and the regular menu styles are used again. The foreground and background colors of the menu items are replaced by the colors from the colorset. If the colorset has a pixmap defined, this pixmap is used as the background of the menu. Note that the *MenuFace* menu style has been optimized for memory consumption and may use less memory than the background from a colorset. The shape mask from the colorset is used to shape the menu. Please refer to the **Colorsets** section for details about colorsets.

ActiveColorset works exactly like *MenuColorset*, but the foreground from the colorset replaces the color given with the *ActiveFore* menu style and the colorset's background color replaces the color given with the *HilightBack* command (to turn on background hilighting you have to use the *HilightBack* menu style too). If specified, the hilight and shadow colors from the colorset are used too. The pixmap and shape mask from the colorset are not used. Hilighting the background or foreground can be turned off individually with the *!ActiveFore* or *!HilightBack* menu styles.

GreyedColorset works exactly like *MenuColorset*, but the foreground from the colorset replaces the color given with the *Greyed* menu style. No other parts of the colorset are used.

TitleColorset works exactly like *MenuColorset*, but is used only for menu titles.

Hilight3DThick, *Hilight3DThin* and *Hilight3DOff* determine if the selected menu item is hilighted with a 3D relief. Thick reliefs are two pixels wide, thin reliefs are one pixel wide.

Hilight3DThickness takes one numeric argument that may be between −50 and +50 pixels. With negative values the menu item gets a pressed in look. The above three commands are equivalent to a thickness of 2, 1 and 0.

Animation and *!Animation* turn menu animation on or off. When animation is on, sub menus that do not fit on the screen cause the parent menu to be shifted to the left so the sub menu can be seen.

Font and *TitleFont* take a font name as an argument. If a font by this name exists it is used for the text of all menu items. If it does not exist or if the name is left blank the built−in default is used. If a *TitleFont* is given, it is used for all menu titles instead of the normal font.

MenuFace enforces a fancy background upon the menus. You can use the same options for *MenuFace* as for the **ButtonStyle**. See description of**ButtonStyle** command and the **Color Gradients** sections for more information. If you use *MenuFace* without arguments the style is reverted back to normal.

Some examples of MenuFaces are:

```
MenuFace DGradient 128 2 lightgrey 50 blue 50 white
MenuFace TiledPixmap texture10.xpm
MenuFace HGradient 128 2 Red 40 Maroon 60 White
MenuFace Solid Maroon
```

Note: The gradient styles H, V, B and D are optimized for high speed and low memory consumption in menus. This is not the case for all the other gradient styles. They may be slow and consume huge amounts of memory, so if you encounter performance problems with them you may be better off by not using them. To improve performance you can try one or all of the following:

Turn hilighting of the active menu item other than foreground color off:

```
MenuStyle <style> Hilight3DOff, !HilightBack
MenuStyle <style> ActiveFore <preferred color>
```

Make sure sub menus do not overlap the parent menu. This can prevent menus being redrawn every time a sub menu pops up or down.

```
MenuStyle <style> PopupOffset 1 100
```

Run your X server with backing storage. If your X Server is started with the −bs option, turn it off. If not try the −wm and +bs options:

```
startx −− −wm +bs
```

You may have to adapt this example to your system (e.g. if you use xinit to start X).

PopupDelay requires one numeric argument. This value is the delay in milliseconds before a sub menu is popped up when the pointer moves over a menu item that has a sub menu. If the value is zero no automatic pop up is done. If the argument is omitted the built−in default is used. Note that the popup delay has no effect if the *PopupImmediately* option is used since sub menus pop up immediately then.

PopupImmediately makes menu items with sub menus pop up it up as soon as the pointer enters the item. The *PopupDelay option* is ignored then. If *PopupDelayed* is used fvwm looks at the *PopupDelay* option if or when this automatic popup happens.

PopdownDelay works exactly like *PopupDelay* but determines the timeout of the *PopupDelayed* style.

PopdownImmediately makes sub menus vanish as soon as the pointer leaves the sub menu and the

correspondent item in the parent menu. With the opposite option *PopdownDelayed* the sub menu only pops down after the time specified with the *PopdownDelay* option. This comes handy when the pointer often strays off the menu item when trying to move into the sub menu. Whenever there is a conflict between the *PopupImmediately*, *PopupDelayed*, *PopupDelay* styles and the *PopdownImmediately*, *PopdownDelayed*, *PopdownDelay* styles, the *Popup...* styles win when using mouse navigation and the *Popdown...* styles win when navigating with the keyboard.

PopupOffset requires two integer arguments. Both values affect where sub menus are placed relative to the parent menu. If both values are zero, the left edge of the sub menu overlaps the left edge of the parent menu. If the first value is non−zero the sub menu is shifted that many pixels to the right (or left if negative). If the second value is non−zero the menu is moved by that many percent of the parent menu's width to the right or left.

PopupActiveArea requires an integer value between 51 and 100. Normally, when the pointer is over a menu item with a sub menu and the pointer enters the area that starts at 75% of the menu width, the sub menu is shown immediately. This percentage can be changed with *PopupActiveArea*. Setting this value to 100 disables this kind of automatic popups altogether. The default value is restored if no or an illegal value is given.

TitleWarp and *!TitleWarp* affect if the pointer warps to the menu title when a sub menu is opened or not. Note that regardless of this setting the pointer is not warped if the menu does not pop up under the pointer.

TitleUnderlines0, *TitleUnderlines1* and *TitleUnderlines2* specify how many lines are drawn below a menu title.

SeparatorsLong and *SeparatorsShort* set the length of menu separators. Long separators run from the left edge all the way to the right edge. Short separators leave a few pixels to the edges of the menu.

TrianglesSolid and *TrianglesRelief* affect how the small triangles for sub menus is drawn. Solid triangles are filled with a color while relief triangles are hollow.

DoubleClickTime requires one numeric argument. This value is the time in milliseconds between two mouse clicks in a menu to be considered as a double click. The default is 450 milliseconds. If the argument is omitted the double click time is reset to this default.

SidePic takes the name of an image file as an argument. The picture is drawn along the left side of the menu. The *SidePic* option can be overridden by a menu specific side pixmap (see **AddToMenu**). If the file name is omitted an existing side pixmap is removed from the menu style.

SideColor takes the name of an X11 color as an argument. This color is used to color the column containing the side picture (see above). The SideColor option can be overridden by a menu specific side color (see **AddToMenu**). If the color name is omitted the side color option is switched off.

PopupAsRootMenu, *PopupAsSubmenu*, *PopupIgnore* and *PopupClose* change the behavior when you click on a menu item that opens a sub menu. With *PopupAsRootMenu* the original menu is closed before the sub menu appears, with *PopupAsSubmenu* it is not, so you can navigate back into the parent menu. Furthermore, with *PopupAsSubmenu* the sub menu is held open (posted) regardless of where you move the mouse. Depending on your menu style this may simplify navigating through the menu. **Any** keystroke while a menu is posted reverts the menu back to the normal behavior. With *PopupClose* the menu is closed when a sub menu item is activated, and the menu stays open if *PopupIgnore* is used (even if the menu was invoked with the **Popup** command). *PopupAsSubmenu* is the default.

RemoveSubmenus instructs fvwm to remove sub menu when you move back into the parent menu. With *HoldSubmenus* the sub menu remains visible. You probably want to use *HoldSubmenus* if you are using the *PopupDelayed* style. *RemoveSubmenus* affects menu navigation with the keyboard.

SelectOnRelease takes an optional key name as an argument. If the given key is released in a menu using this style, the current menu item is selected. This is intended for Alt-Tab **WindowList** navigation. The key name is a standard X11 key name as defined in */usr/include/X11/keysymdef.h*, (without the *XK_* prefix), or the keysym database */usr/X11R6/lib/X11/XKeysymDB*. To disable this behavior, omit the key name.

Note: Some X servers do not support KeyRelease events. *SelectOnRelease* does not work on such a machine.

ItemFormat takes a special string as its argument that determines the layout of the menu items. Think of the format string as if it were a menu item. All you have to do is tell fvwm where to place the different parts of the menu item (i.e. the labels, the triangle denoting a sub menu, the mini icons and the side pic) in the blank area. The string consists of spaces, Tab characters and formatting directives beginning with '%'. Any illegal characters and formatting directives are silently ignored:

%l, **%c** and **%r**
Insert the next item label. Up to three labels can be used. The item column is left−aligned (**%l**), centered (**%c**) or right−aligned (**%r**).

%i
Inserts the mini icon.

%> and **%<**
Insert the sub menu triangle pointing either to the right (**%>**) or to the left (**%<**).

%|
The first **%|** denotes the beginning of the area that is highlighted either with a background color or a relief (or both). The second **%|** marks the end of this area. **%|** can be used up to twice in the string. If you do not add one or both of them, fvwm sets the margins to the margins of the whole item (not counting the side picture).

%s
Places the side picture either at the beginning or the end of the menu. This directive may be used only once and only as the first or last in the format string. If the **%s** is not at the beginning of the string, menus are not drawn properly.

Space, **Tab**, **%Space** and **%Tab**
Add gap of one space, or a tab, using the width of the menu font. When using a tab, the size of the gap can be one to 8 spaces since the tab position is a multiple of 8 from the edge of the menu. The whole string must be quoted if spaces or tabs are used.

%p
Like Space and Tab **%p** inserts an empty area into the item, but with better control of its size (see below).

You can define an additional space before and after each of the objects like this:

%*left***.***right***p**

This means: if the object is defined in the menu (e.g. if it is **%s** and you use a side picture, or it is **%l** for the third column and there are items defined that actually have a third column), then add *left* pixels before the object and *right* pixels after it. You may leave out the *left* or the *.right* parts if you do not need them. All values up to the screen width are allowed. Even negative values can be used with care. The **p** may be replaced with any other formatting directives described above.

Note: Only items defined in the format string are visible in the menus. So if you do not put a **%s** in there you do not see a side picture, even if one is specified.

Note: The *SubmenusLeft* style changes the default *ItemFormat* string, but if it was set manually it is not modified.

Note: If any unformatted title of the menu is wider than the widest menu item, the spaces between

the different parts of the menu items are enlarged to match the width of the title. Leading left aligned objects in the format string (**%l**, **%i**, **%<**, first **%|**) stick to the left edge of the menu and trailing right aligned objects (**%r**, **%i**, **%>**, second **%|**) stick to the right edge. The gaps between the remaining items are enlarged equally.

Examples:

```
MenuStyle * ItemFormat "%.4s%.1|%.5i%.5l%.5l%.5r%.5i%2.3>%1|"
```

Is the default string used by fvwm: (side picture + 4 pixels gap) (beginning of the hilighted area + 1 pixel gap) (mini icon + 5p) (first column left aligned + 5p) (second column left aligned + 5p) (third column right aligned + 5p) (second mini icon + 5p) (2p + sub menu triangle + 3p) (1p + end of hilighted area).

```
MenuStyle * ItemFormat "%.1|%3.2<%5i%5l%5l%5r%5i%1|%4s"
```

Is used by fvwm with the *SubmenusLeft* option below.

VerticalItemSpacing and *VerticalTitleSpacing* control the vertical spacing of menu items and titles like *ItemFormat* controls the horizontal spacing. Both take two numeric arguments that may range from −100 to +100. The first is the gap in pixels above a normal menu item (or a menu title), the second is the gap in pixels below it. Negative numbers do not make much sense and may screw up the menu completely. If no arguments are given or the given arguments are invalid, the built−in defaults are used: one pixel above the item or title and two below.

VerticalMargins can be used to add some padding at the top and bottom of menus. It takes two numeric arguments that must be positive integers (or zero). If the number of arguments or its values are incorrect, fvwm defaults both to 0, which means no padding at all. If the values are correct, the first one is used for the top margin, and the second one is used for the bottom margin.

SubmenusLeft mirrors the menu layout and behavior. Sub menus pop up to the left, the sub menu triangle is drawn left and the mini icon and side picture are drawn at the right side of the menu. The default is *SubmenusRight*. The position hints of a menu are also affected by this setting, i.e. position hints using *item* or *menu* as context rectangle and position hints using *m* offsets.

AutomaticHotkeys and *!AutomaticHotkeys* control the menu's ability to automatically provide hot−keys on the first character of each menu item's label. This behavior is always overridden if an explicit hot−key is assigned in the **AddToMenu** command.

UniqueHotkeyActivatesImmediate and *!UniqueHotkeyActivatesImmediate* controls how menu items are invoked when used with hotkeys. By default, if a given menu entry only has one completeable match for a given hotkey, the action for that menu entry is invoked and the menu is closed. This is due to the*UniqueHotk eyActivatesImmediate* option. However, the menu can be told to remain open, waiting for the user to invoke the selected item instead when there is only one matched item for a given hotkey, by using the *!UniqueHotkeyActivatesImmediate* option.

MouseWheel controls the ability to scroll the menu using a mouse wheel. It takes one argument, that can be one of ScrollsPointer, ScrollsMenu, ScrollsMenuBackwards or ActivatesItem. ScrollsPointer makes the mouse wheel scroll the pointer over a menu. This is the default. ScrollsMenu and ScrollsMenuBackwards scroll the menu beneath the pointer. ActivatesItem disables scrolling by mouse wheel and makes the use of a mouse wheel act as if the menu was clicked. If no argument is supplied the default setting is restored.

ScrollOffPage allows a menu to be scrolled out of the visible area if *MouseWheel* is set to ScrollsMenu or ScrollsMenuBackwards. This is the default. The opposite,*!Scr ollOffPage* disables this behaviour.

TrianglesUseFore draws sub menu triangles with the foreground color of the menu colorset (normally drawn with the hilight color). *!TrianglesUseFore* disables this behaviour.

Examples:

```
MenuStyle * Mwm
MenuStyle * Foreground Black, Background gray40
MenuStyle * Greyed gray70, ActiveFore White
MenuStyle * !HilightBack, Hilight3DOff
MenuStyle * Font lucidasanstypewriter−14
MenuStyle * MenuFace DGradient 64 darkgray MidnightBlue

MenuStyle red Mwm
MenuStyle red Foreground Yellow
MenuStyle red Background Maroon
MenuStyle red Greyed Red, ActiveFore Red
MenuStyle red !HilightBack, Hilight3DOff
MenuStyle red Font lucidasanstypewriter−12
MenuStyle red MenuFace DGradient 64 Red Black
```

Note that all style options could be placed on a single line for each style name.

MenuStyle *forecolor backcolor shadecolor font style* [*anim*]
This is the old syntax of the **MenuStyle** command. It is obsolete and may be removed in the future. Please use the new syntax as described above.

Sets the menu style. When using monochrome the colors are ignored. The *shadecolor* is the one used to draw a menu−selection which is prohibited (or not recommended) by the Mwm hints which an application has specified. The style option is either *Fvwm*, *Mwm* or *Win*, which changes the appearance and operation of the menus.

Mwm and *Win* style menus popup sub menus automatically. *Win* menus indicate the current menu item by changing the background to black. *Fvwm* sub menus overlap the parent menu, *Mwm* and *Win* style menus never overlap the parent menu.

When the *anim* option is given, sub menus that do not fit on the screen cause the parent menu to be shifted to the left so the sub menu can be seen. See also **SetAnimation** command.

Popup *PopupName* [*position*] [*default−action*]
This command has two purposes: to bind a menu to a key or mouse button, and to bind a sub menu into a menu. The formats for the two purposes differ slightly. The*position* ar guments are the same as for **Menu**. The command*default−action* is in voked if the user clicks a button to invoke the menu and releases it immediately again (or hits the key rapidly twice if the menu is bound to a key). If the default action is not specified, double clicking on the menu does nothing. However, if the menu begins with a menu item (i.e. not with a title or a separator) and the default action is not given, double clicking invokes the first item of the menu (but only if the pointer really was over the item).

To bind a previously defined pop−up menu to a key or mouse button:

The following example binds mouse buttons 2 and 3 to a pop−up called "Window Ops". The menu pops up if the buttons 2 or 3 are pressed in the window frame, side−bar, or title−bar, with no modifiers (none of shift, control, or meta).

```
Mouse 2 FST N Popup "Window Ops"
Mouse 3 FST N Popup "Window Ops"
```

Pop−ups can be bound to keys through the use of the **Key** command. Pop−ups can be operated without using the mouse by binding to keys and operating via the up arrow, down arrow, and enter keys.

To bind a previously defined pop−up menu to another menu, for use as a sub menu:

The following example defines a sub menu "Quit−Verify" and binds it into a main menu, called "RootMenu":

```
AddToMenu Quit−Verify
 + "Really Quit Fvwm?" Title
 + "Yes, Really Quit"  Quit
 + "Restart Fvwm"      Restart
 + "Restart Fvwm 1.xx" Restart fvwm1 −s
 + ""                  Nop
 + "No, Don't Quit"    Nop

AddToMenu RootMenu "Root Menu" Title
 + "Open XTerm Window" Popup NewWindowMenu
 + "Login as Root"     Exec exec xterm −T Root −n Root −e su −
 + "Login as Anyone"   Popup AnyoneMenu
 + "Remote Hosts"      Popup HostMenu
 + ""                  Nop
 + "X utilities"       Popup Xutils
 + ""                  Nop
 + "Fvwm Modules"      Popup Module−Popup
 + "Fvwm Window Ops"   Popup Window−Ops
 + ""                  Nop
 + "Previous Focus"    Prev (AcceptsFocus) Focus
 + "Next Focus"        Next (AcceptsFocus) Focus
 + ""                  Nop
 + "Refresh screen"    Refresh
 + ""                  Nop
 + "Reset X defaults"  Exec xrdb −load \
                       $HOME/.Xdefaults
 + ""                  Nop
 + ""                  Nop
 + Quit                Popup Quit−Verify
```

Popup differs from **Menu** in that pop−ups do not stay up if the user simply clicks. These are popup−menus, which are a little hard on the wrist. **Menu** menus stay up on a click action. See the **Menu** command for an explanation of the interactive behavior of menus. A menu can be open up to ten times at once, so a menu may even use itself or any of its predecessors as a sub menu.

TearMenuOff

When assigned to a menu item, it inserts a tear off bar into the menu (a horizontal broken line). Activating that item tears off the menu. If the menu item has a label, it is shown instead of the broken line. If used outside menus, this command does nothing. Examples:

```
AddToMenu WindowMenu
+ I "" TearMenuOff

AddToMenu RootMenu
+ I "click here to tear me off" TearMenuOff
```

Title

Does nothing This is used to insert a title line in a popup or menu.

Miscellaneous Commands

BugOpts [*option* [*bool*]], ...

This command controls several workarounds for bugs in third party programs. The individual options are separated by commas. The optional argument *bool* is a boolean argument and controls if the bug workaround is enabled or not. It can either be "True" or "False" to turn the option on or off, or "toggle" to switch is back and forth. If *bool* is omitted, the default setting is restored.

FlickeringMoveWorkaround disables ConfigureNotify events that are usually sent to an application

while it is moved. If some windows flicker annoyingly while being moved, this option may help you. Note that if this problem occurs it is not an fvwm bug, it is a problem of the application.

MixedVisualWorkaround makes fvwm install the root colormap before it does some operations using the root window visuals. This is only useful when the **−visual** option is used to start fvwm and then only with some configurations of some servers (e.g. Exceed 6.0 with an 8 bit PseudoColor root and fvwm using a 24 bit TrueColor visual).

The *ModalityIsEvil* option controls whether Motif applications have the ability to have modal dialogs (dialogs that force you to close them first before you can do anything else). The default is to not allow applications to have modal dialogs. Use this option with care. Once this option is turned on, you have to restart fvwm to turn it off.

RaiseOverNativeWindows makes fvwm try to raise the windows it manages over native windows of the X server's host system. This is needed for some X servers running under Windows, Windows NT or Mac OS X. Fvwm tries to detect if it is running under such an X server and initializes the flag accordingly.

RaiseOverUnmanaged makes fvwm try to raise the windows it manages over override_redirect windows. This is used to cope with ill−mannered applications that use long−lived windows of this sort, contrary to ICCCM conventions. It is useful with the*Unmana ged* style option too.

FlickeringQtDialogsWorkaround suppresses flickering of the focused window in some modules when using KDE or QT applications with application modal dialog windows. By default this option is turned on. This option may be visually disturbing for other applications using windows not managed by fvwm. Since these applications are rare it is most likely safe to leave this option at its default.

QtDragnDropWorkaround suppresses the forwarding of unknown ClientEvent messages to windows −− usually this is harmless, but Qt has problems handling unrecognised ClientEvent messages. Enabling this option might therefore help for Qt applications using DragnDrop. This option is off by default.

EWMHIconicStateWorkaround is needed by EWMH compliant pagers or taskbars which represent windows which are on a different desktops as iconified. These pagers and taskbars use a version of the EWMH specification before version 1.2 (the current KDE 2 & 3 versions). These pagers and taskbars use the IconicState WM_STATE state to determine if an application is iconified. This state, according to the ICCCM, does not imply that a window is iconified (in the usual sense). Turning on this option forces fvwm to establish an equivalence between the IconicState WM_STATE state and the iconified window. This violates ICCCM compliance but should not cause big problems. By default this option is off.

With the *DisplayNewWindowNames* enabled, fvwm prints the name, icon name (if available), resource and class of new windows to the console. This can help in finding the correct strings to use in the **Style** command.

When the *ExplainWindowPlacement* option is enabled, fvwm prints a message to the console whenever a new window is placed or one of the commands **PlaceAgain**, **Recapture** or **RecaptureWindow** is used. The message explains on which desk, page, Xinerama screen and position it was placed and why. This option can be used to figure out why a specific window does not appear where you think it should.

The *DebugCRMotionMethod* option enables some debugging code in the ConfigureRequest handling routines of fvwm. It is not helpful for the user, but if you report a bug to the fvwm team we may ask you to enable this option.

The *TransliterateUtf8* option enables transliteration during conversions from utf−8 strings. By default fvwm will not transliterate during conversion, but will fall back to alternate strings provided by the clients if conversion from utf−8 fails due to characters which have no direct correspondence in the target charecter set. Some clients however neglect to set non utf−8 properties correctly in which case this option may help.

BusyCursor [*Option bool*], ...

This command controls the cursor during the execution of certain commands. *Option* can be *DynamicMenu*, *ModuleSynchronous*, *Read*, *Wait* or *. An option must be followed by a boolean argument *bool*. You can use commas to separate individual options. If you set an option to "True", then when the corresponding command is run, fvwm displays the cursor of the *WAIT* context of the **CursorStyle** command. "False" forces to not display the cursor. The default is:

```
BusyCursor DynamicMenu False, ModuleSynchronous False, \
  Read False, Wait False
```

The * option refers to all available options.

The *Read* option controls the **PipeRead** command.

The *DynamicMenu* option affects the *DynamicPopupAction* and *MissingSubmenuFunction* options of the **AddToMenu** command. If this option is set to "False", then the busy cursor is not displayed during a dynamic menu command even if this command is a **Read** or **PipeRead** command and the *Read* option is set to "True".

The *ModuleSynchronous* option affects the **ModuleSynchronous** command. If this option is set to "False", then the busy cursor is not displayed while fvwm waits for a module started by **ModuleSynchronous** to complete its startup.

The *Wait* option affects only the root cursor. During a wait pause the root cursor is replaced by the busy cursor and fvwm is still fully functional (you can escape from the pause, see the **EscapeFunc** command). If you want to use this option and if you do not use the default root cursor, you must set your root cursor with the **CursorStyle** command.

ClickTime [*delay*]

Specifies the maximum delay in milliseconds between a button press and a button release for the **Function** command to consider the action a mouse click. The default delay is 150 milliseconds. Omitting the delay value resets the **ClickTime** to the default.

ColorLimit *limit*

This command is obsolete. See the **−−color−limit** option to fvwm.

ColormapFocus FollowsMouse | FollowsFocus

By default, fvwm installs the colormap of the window that the cursor is in. If you use

```
ColormapFocus FollowsFocus
```

then the installed colormap is the one for the window that currently has the keyboard focus.

CursorStyle *context* [*num* | *name* | None | Tiny | *file* [*x y*] [*fg bg*]]

Defines a new cursor for the specified context. Note that this command can not control the shapes an applications uses, for example, to indicate that it is busy. The various contexts are:

POSITION (top_left_corner)
: used when initially placing windows

TITLE (top_left_arrow)
: used in a window title−bar

DEFAULT (top_left_arrow)
: used in windows that do not set their cursor

SYS (hand2)
: used in one of the title−bar buttons

MOVE (fleur)
: used when moving or resizing windows

RESIZE (sizing)
: used when moving or resizing windows

WAIT (watch)
used during certain fvwm commands (see **BusyCursor** for details)

MENU (top_left_arrow)
used in menus

SELECT (crosshair)
used when the user is required to select a window

DESTROY (pirate)
used for **Destroy**, **Close**, and **Delete** commands

TOP (top_side)
used in the top side−bar of a window

RIGHT (right_side)
used in the right side−bar of a window

BOTTOM (bottom_side)
used in the bottom side−bar of a window

LEFT (left_side)
used in the left side−bar of a window

TOP_LEFT (top_left_corner)
used in the top left corner of a window

TOP_RIGHT (top_right_corner)
used in the top right corner of a window

BOTTOM_LEFT (bottom_left_corner)
used in the bottom left corner of a window

BOTTOM_RIGHT (bottom_right_corner)
used in the bottom right corner of a window

TOP_EDGE (top_side)
used at the top edge of the screen

RIGHT_EDGE (right_side)
used at the right edge of the screen

BOTTOM_EDGE (bottom_side)
used at the bottom edge of the screen

LEFT_EDGE (left_side)
used at the left edge of the screen

ROOT (left_ptr)
used as the root cursor

STROKE (plus)
used during a **StrokeFunc** command.

The defaults are shown in parentheses above. If you ever want to restore the default cursor for a specific context you can omit the second argument.

The second argument is either the numeric value of the cursor as defined in the include file *X11/cursorfont.h* or its name (without the XC_ prefix). Alternatively, the xpm file name may be specified. The name can also be*None* (no cursor) or *T iny* (a single pixel as the cursor).

```
# make the kill cursor be XC_gumby (both forms work):
CursorStyle DESTROY 56
CursorStyle DESTROY gumby
```

Alternatively, the cursor can be loaded from an (XPM, PNG or SVG) image *file*. If fvwm is

compiled with Xcursor support, full ARGB is used, and (possibly animated) cursor files made with the **xcursorgen** program can be loaded. Otherwise the cursor is converted to monochrome.

The optional *x* and *y* arguments (following a *file* argument) specifies the hot−spot coordinate with 0 0 as the top left corner of the image. Coordinates within the image boundary are valid and overrides any hot−spot defined in the (XPM/Xcursor) image file. An invalid or undefined hot−spot is placed in the center of the image.

```
CursorStyle ROOT cursor_image.png 0 0
```

The optional *fg* and *bg* arguments specify the foreground and background colors for the cursor, defaulting to black and white (reverse video compared to the actual bitmap). These colors are only used with monochrome cursors. Otherwise they are silently ignored.

```
CursorStyle ROOT nice_arrow.xpm yellow black
```

DefaultColors [*foreground*] [*background*]

DefaultColors sets the default foreground and background colors used in miscellaneous windows created by fvwm, for example in the geometry feedback windows during a move or resize operation. If you do not want to change one color or the other, use − as its color name. To revert to the built−in default colors omit both color names. Note that the default colors are not used in menus, window titles or icon titles.

DefaultColorset [*num*]

DefaultColorset sets the colorset used by the windows controlled by the **DefaultColors** command. To revert back to the **DefaultColors** colors use

```
DefaultColorset −1
```

or any variant of the **DefaultColors** command.

DefaultFont [*fontname*]

DefaultFont sets the default font to font *fontname*. The default font is used by fvwm whenever no other font has been specified. To reset the default font to the built−in default, omit the argument. The default font is used for menus, window titles, icon titles as well as the geometry feedback windows during a move or resize operation. To override the default font in a specific context, use the **Style** * *Font*, **Style** * *IconFont*, or **MenuStyle** commands.

DefaultIcon *filename*

Sets the default icon which is used if a window has neither an client−supplied icon nor an icon supplied via the *Icon* option of the **Style** command.

DefaultLayers *bottom put top*

Changes the layers that are used for the *StaysOnBottom*, *StaysPut*, *StaysOnTop* **Style** options. Initially, the layers 2, 4 and 6 are used.

Deschedule [*command_id*]

Removes all commands that were scheduled with the id *command_id* with the **Schedule** command from the list of commands to be executed unless they were already executed. If the*command_id* is omitted, the value of the variable $[schedule.last] is used as the id.

Emulate Fvwm | Mwm | Win

This command is a catch all for how miscellaneous things are done by fvwm. Right now this command affects where the move/resize feedback window appears and how window placement is aborted. To have more Mwm− or Win−like behavior you can call **Emulate** with *Mwm* or *Win* as its argument. With Mwm resize and move feedback windows are in the center of the screen, instead of the upper left corner. This also affects how manual placement is aborted. See the *ManualPlacement* description.

EscapeFunc

By default the key sequence Ctrl-Alt-Escape allows for escaping from a **Wait** pause and from a locked **ModuleSynchronous** command. The **EscapeFunc** command used with the **Key** command allows for configuring this key sequence. An example:

Key Escape A MC −
Key Escape A S EscapeFunc

replaces the Ctrl-Alt-Escape key sequence with Shift-Escape for aborting a **Wait** pause and **ModuleSynchronous** command. **EscapeFunc** used outside the **Key** command does nothing.

FakeClick [*command value*] ...

This command is mainly intended for debugging fvwm and no guarantees are made that it works for you. *FakeClick* can simulate mouse button press and release events and pass them to fvwm or the applications. The parameters are a list of commands which consist of pairs of *command* tokens and integer *values*, The *press* and *release* commands are followed by the appropriate mouse button number and generate a button press or release event on the window below the pointer. The *wait* commands pauses fvwm for the given number of milliseconds. The *modifiers* command simulates pressing or releasing modifier keys. The values 1 to 5 are mapped to Mod1 to Mod5 while 6, 7 and 8 are mapped to Shift , Lock and Control The modifier is set for any further button events. To release a modifier key, use the corresponding negative number. The*depth* command determines to which window the button events are sent. With a depth of 1, all events go to the root window, regardless of the pointer's position. With 2, the event is passed to the top level window under the pointer which is usually the frame window. With 3, events go to the client window. Higher numbers go to successive sub windows. Zero (0) goes to the smallest window that contains the pointer. Note that events propagate upward.

FakeClick depth 2 press 1 wait 250 release 1

This simulates a click with button 1 in the parent window (depth 2) with a delay of 250 milliseconds between the press and the release. Note: all command names can be abbreviated with their first letter.

FakeKeypress [*command value*] ...

This command is mainly intended for debugging fvwm and no guarantees are made that it works for you. **FakeKeypress** can simulate key press and release events and pass them to fvwm or applications. The parameters are a list of commands which consist of pairs of command tokens and values. The*pr ess* and *release* commands are followed by a key name. The key name is a standard X11 key name as defined in */usr/include/X11/keysymdef.h*, (without the *XK_* prefix), or the keysym database */usr/X11R6/lib/X11/XKeysymDB*. The*wait*, *modifier s* and *depth* commands are the same as those used by **FakeClick**.

Save all GVim sessions with: "Esc:w\n"

All (gvim) FakeKeypress press Escape \
press colon \
press w \
press Return

Save & exit all GVim sessions with: "Esc:wq\n"

All (gvim) FakeKeypress press Escape \
press colon \
press w \
press q \
press Return

Send A to a specific window:

WindowId 0x3800002 FakeKeypress press A

Note: all command names can be abbreviated with their first letter.

GlobalOpts [*options*]

This command is obsolete. Please replace the global options in your configuration file according to the following table:

GlobalOpts *WindowShadeShrinks*
−−>
Style * *WindowShadeShrinks*

GlobalOpts *WindowShadeScrolls*
−−>
Style * *WindowShadeScrolls*

GlobalOpts *SmartPlacementIsReallySmart*
−−>
Style * *MinOverlapPlacement*

GlobalOpts *SmartPlacementIsNormal*
−−>
Style * *TileCascadePlacement*

GlobalOpts *ClickToFocusDoesntPassClick*
−−>
Style * *ClickToFocusPassesClickOff*

GlobalOpts *ClickToFocusPassesClick*
−−>
Style * *ClickToFocusPassesClick*

GlobalOpts *ClickToFocusDoesntRaise*
−−>
Style * *ClickToFocusRaisesOff*

GlobalOpts *ClickToFocusRaises*
−−>
Style * *ClickToFocusRaises*

GlobalOpts *MouseFocusClickDoesntRaise*
−−>
Style * *MouseFocusClickRaisesOff*

GlobalOpts *MouseFocusClickRaises*
−−>
Style * *MouseFocusClickRaises*

GlobalOpts *NoStipledTitles*
−−>
Style * !*StippledTitle*

GlobalOpts *StipledTitles*
−−>
Style * *StippledTitle*

GlobalOpts *CaptureHonorsStartsOnPage*
−−>
Style * *CaptureHonorsStartsOnPage*

GlobalOpts *CaptureIgnoresStartsOnPage*
−−>
Style * *CaptureIgnoresStartsOnPage*

GlobalOpts *RecaptureHonorsStartsOnPage*
−−>
Style * *RecaptureHonorsStartsOnPage*

GlobalOpts *RecaptureIgnoresStartsOnPage*
−−>
Style * *RecaptureIgnoresStartsOnPage*

GlobalOpts *ActivePlacementHonorsStartsOnPage*
−−>
Style * *ManualPlacementHonorsStartsOnPage*

GlobalOpts *ActivePlacementIgnoresStartsOnPage*
−−>
Style * *ManualPlacementIgnoresStartsOnPage*

GlobalOpts *RaiseOverNativeWindows*
−−>
BugOpts *RaiseOverNativeWindows* on

GlobalOpts *IgnoreNativeWindows*
−−>
BugOpts *RaiseOverNativeWindows* off

HilightColor *textcolor backgroundcolor*
This command is obsoleted by the **Style** options *HilightFore* and *HilightBack*. Please use

Style * *HilightFore* textcolor, *HilightBack* backgroundcolor

instead.

HilightColorset [*num*]
This command is obsoleted by the **Style** option *HilightColorset*. Please use

Style * *HilightColorset* num

instead.

IconFont [*fontname*]
This command is obsoleted by the **Style** option *IconFont*. Please use

Style * IconFont fontname

instead.

IconPath *path*
This command is obsolete. Please use **ImagePath** instead.

ImagePath *path*

Specifies a colon separated list of directories in which to search for images (both monochrome and pixmap). To find an image given by a relative pathname, fvwm looks into each directory listed in turn, and uses the first file found.

If a directory is given in the form "/some/dir;.ext", this means all images in this directory have the extension ".ext" that should be forced. The original image name (that may contain another extension or no extension at all) is not probed, instead ".ext" is added or replaces the original extension. This is useful, for example, if a user has some image directories with ".xpm" images and other image directories with the same names, but ".png" images.

The *path* may contain environment variables such as *$HOME* (or *${HOME}*). Further, a '+' in the *path* is expanded to the previous value of the path, allowing appending or prepending to the path easily.

For example:

```
ImagePath $HOME/icons:+:/usr/include/X11/bitmaps
```

Note: if the **FvwmM4** module is used to parse your *config* files, then m4 may want to mangle the word "include" which frequently shows up in the **ImagePath** command. To fix this one may add

```
undefine(‘include’)
```

prior to the **ImagePath** command, or better: use the **−m4−prefix** option to force all **m4** directives to have a prefix of "m4_" (see the **FvwmM4** man page).

LocalePath *path*

Specifies a colon separated list of "locale path" in which to search for string translations. A locale path is constituted by a directory path and a text domain separated by a semicolon (';'). As an example the default locale path is:

```
/install_prefix/share/locale;fvwm
```

where install_prefix is the fvwm installation directory. With such a locale path translations are searched for in

```
/install_prefix/share/locale/lang/LC_MESSAGES/fvwm.mo
```

where *lang* depends on the locale. If no directory is given the default directory path is assumed. If no text domain is given, **fvwm** is assumed. Without argument the default locale path is restored.

As for the **ImagePath** command, *path* may contain environment variables and a '+' to append or prepend the locale path easily.

For example, the fvwm−themes package uses

```
LocalePath ";fvwm−themes:+"
```

to add locale catalogs.

The default fvwm catalog contains a few strings used by the fvwm executable itself (Desk and Geometry) and strings used in some default configuration files and **FvwmForm** configuration. You can take a look at the po/ subdirectory of the fvwm source to get the list of the strings with a possible translation in various languages. At present, very few languages are supported.

The main use of locale catalogs is via the "$[gt.string]" parameter:

```
DestroyMenu MenuFvwmWindowOps
AddToMenu   MenuFvwmWindowOps "$[gt.Window Ops]" Title
+ "$[gt.&Move]"          Move
+ "$[gt.&Resize]"        Resize
```

```
+ "$[gt.R&aise]"          Raise
+ "$[gt.&Lower]"          Lower
+ "$[gt.(De)&Iconify]"    Iconify
+ "$[gt.(Un)&Stick]"      Stick
+ "$[gt.(Un)Ma&ximize]"   Maximize
+ "" Nop
+ "$[gt.&Close]"          Close
+ "$[gt.&Destroy]"        Destroy
```

gives a menu in the locale languages if translations are available.

Note that the **FvwmScript** module has a set of special instructions for string translation. It is out of the scope of this discussion to explain how to build locale catalogs. Please refer to the GNU gettext documentation.

PixmapPath *path*

This command is obsolete. Please use **ImagePath** instead.

PrintInfo *subject* [*verbose*]

Print information on *subject* on stderr. An optional integer argument *verbose* defines the level of information which is given. The current valid subjects are:

Colors which prints information about the colors used by fvwm. This useful on screens which can only display 256 (or less) colors at once. If *verbose* is one or greater the palette used by fvwm is printed. If you have a limited color palette, and you run out of colors, this command might be helpful.

ImageCache which prints information about the images loaded by fvwm. If *verbose* is one or greater all images in the cache will be listed together with their respective reuse.

Locale which prints information on your locale and the fonts that fvwm used. *verbose* can be 1 or 2.

nls which prints information on the locale catalogs that fvwm used

style which prints information on fvwm styles. *verbose* can be 1.

bindings which prints information on all the bindings fvwm has: key, mouse and stroke bindings. *verbose* has no effect with this option.

infostore which prints information on all entries in the infostore, listing the key and its value. *verbose* has no effect with this option.

Repeat

When the **Repeat** command is invoked, the last command that was executed by fvwm is executed again. This happens regardless of whether it was triggered by user interaction, a module or by an X event. Commands that are executed from a function defined with the **Function** command, from the **Read** or **PipeRead** commands or by a menu are not repeated. Instead, the function, menu or the **Read** or **PipeRead** command is executed again.

Schedule [Periodic] *delay_ms* [*command_id*] *command*

The *command* is executed after about *delay_ms* milliseconds. This may be useful in some tricky setups. The*command* is e xecuted in the same context window as the **Schedule** command. An optional integer argument *command_id* may be given in decimal, hexadecimal or octal format. This id can be used with the **Deschedule** command to remove the scheduled command before it is executed. If no id is given, fvwm uses negative id numbers, starting with −1 and decreasing by one with each use of the **Schedule** command. Note that the **Schedule** command and its arguments undergo the usual command line expansion, and, when *command* is finally executed, it is expanded again. It may therefore be necessary to quote the parts of the command that must not be expanded twice.

Note: A window's id as it is returned with $[w.id] can be used as the *command_id*. Example:

Current Schedule 1000 $[w.id] **WindowShade**

The **Schedule** command also supports the optional keyword *Periodic* which indicates that the *command* should be executed every *delay_ms*. Example:

Schedule Periodic 10000 **PipeRead** '[−N "$MAIL"] && echo \
Echo You have mail'

Use the **Deschedule** command to stop periodic commands.

State *state* [*bool*]
Sets, clears or toggles one of the 32 user defined states which are associated with each window. The *state* is a number ranging from 0 to 31. The states have no meaning in fvwm, but they can be checked in conditional commands like **Next** with the *State* condition. The optional argument *bool* is a boolean argument. "True" sets the given state, while "False" clears it. Using "toggle" switches to the opposite state. If the *bool* argument is not given, the state is toggled.

WindowFont [*fontname*]
This command is obsoleted by the **Style** option *Font*. Please use

Style * *Font* fontname

instead.

WindowList [(*conditions*)] [*position*] [*options*] [*double−click−action*]
Generates a pop−up menu (and pops it up) in which the title and geometry of each of the windows currently on the desktop are shown.

The format of the geometry part is: *desk*(*layer*): *x−geometry sticky*, where *desk* and *layer* are the corresponding numbers and *sticky* is empty or a capital S. The geometry of iconified windows is shown in parentheses. Selecting an item from the window list pop−up menu causes the interpreted function "WindowListFunc" to be run with the window id of that window passed in as $0. The default "WindowListFunc" looks like this:

AddToFunc WindowListFunc
+ I **Iconify** off
+ I **FlipFocus**
+ I **Raise**
+ I **WarpToWindow** 5p 5p

You can destroy the built−in "WindowListFunc" and create your own if these defaults do not suit you.

The window list menu uses the "WindowList" menu style if it is defined (see **MenuStyle** command). Otherwise the default menu style is used. To switch back to the default menu style, issue the command

DestroyMenuStyle WindowList

Example:

MenuStyle WindowList *SelectOnRelease* Meta_L

The *conditions* can be used to exclude certain windows from the window list. Please refer to the **Current** command for details. Only windows that match the given conditions are displayed in the window list. The *options* below work vice versa: windows that would otherwise not be included in the window list can be selected with them. The *conditions* always override the *options*.

The *position* arguments are the same as for **Menu**. The command*double−clic k−action* is invoked if the user double−clicks (or hits the key rapidly twice if the menu is bound to a key) when bringing the window list. The *double−click−action* must be quoted if it consists of more than one word.

The *double−click−action* is useful to define a default window if you have bound the window list to a key (or button) like this:

```
# Here we call an existing function, but
# it may be different.  See the default
# WindowListFunc definition earlier in this
# man page.
AddToFunc SwitchToWindow
+ I WindowListFunc

Key Tab A M WindowList "Prev SwitchToWindow"
```

Hitting Alt-Tab once it brings up the window list, if you hit it twice the focus is flipped between the current and the last focused window. With the proper *SelectOnRelease* menu style (see example above) a window is selected as soon as you release the Alt key.

The *options* passed to WindowList are separated by commas and can be *Geometry* / *NoGeometry* / *NoGeometryWithInfo*, *NoDeskNum*, *NoLayer*, *NoNumInDeskTitle*, *NoCurrentDeskTitle*, *MaxLabelWidth width*, *TitleForAllDesks*, *Function funcname*, *Desk desknum*, *CurrentDesk*, *NoIcons* / *Icons* / *OnlyIcons*, *NoNormal* / *Normal* / *OnlyNormal*, *NoSticky* / *Sticky* / *OnlySticky*, *NoStickyAcrossPages* / *StickyAcrossPages* / *OnlyStickyAcrossPages*, *NoStickyAcrossDesks* / *StickyAcrossDesks* / *OnlyStickyAcrossDesks*, *NoOnTop* / *OnTop* / *OnlyOnTop*, *NoOnBottom* / *OnBottom* / *OnlyOnBottom*, *Layer m [n]*, *UseSkipList* / *OnlySkipList*, *NoDeskSort*, *ReverseOrder*, *CurrentAtEnd*, *IconifiedAtEnd*, *UseIconName*, *Alphabetic* / *NotAlphabetic*, *SortByResource*, *SortByClass*, *NoHotkeys*, *SelectOnRelease*.

(Note − normal means not iconic, sticky, or on top)

With the *SortByResource* option windows are alphabetically sorted first by resource class, then by resource name and then by window name (or icon name if *UseIconName* is specified). *ReverseOrder* also works in the expected manner.

With the *SortByClass* option windows are sorted just like with *SortByResource*, but the resource name is not taken into account, only the resource class.

The *SelectOnRelease* option works exactly like the **MenuStyle** option with the same name, but overrides the option given in a menu style. By default, this option is set to the left Alt key. To switch it off, use *SelectOnRelease* without a key name.

If you pass in a function via **Function** funcname, it is called within a window context of the selected window:

```
AddToFunc IFunc I Iconify toggle
WindowList Function IFunc, NoSticky, CurrentDesk, NoIcons
```

If you use the *Layer m* [*n*] option, only windows in layers between m and n are displayed. n defaults to m. With the *ReverseOrder* option the order of the windows in the list is reversed.

With the *CurrentAtEnd* option the currently focused window (if any) is shown at the bottom of the list. This is mostly intended for simulating the Alt−Tab behavior in another GUI.

IconifiedAtEnd makes iconified windows be moved to the end of the list. This is also from another GUI.

The *NoGeometry* option causes fvwm to not display the geometries as well as the separators which indicate the different desktops. *NoGeometryWithInfo* removes the geometries, but keep the desktop information and indicates iconic windows. *NoDeskNum* causes fvwm to not display the desktop number in the geometry or before the window title with the *NoGeometryWithInfo* option. *NoNumInDeskTitle* is only useful if a desktop name is defined with the **DesktopName** command. It causes fvwm to not display the desktop number before the desktop name. By default, the WindowList menu have a title which indicates the current desk or the selected desktop if the *Desk*

condition is used. The *NoCurrentDeskTitle* option removes this title. *TitleForAllDesks* causes fvwm to add a menu title with the desk name and/or number before each group of windows on the same desk. With *NoLayer*, the layer of the window is not diplayed. The options *ShowPage*, *ShowPageX* and *ShowPageY* enable displaying the page of the window rounded multiples of the display size. With *ShowScreen*, the window's Xinerama screen number is displayed.

The *MaxLabelWidth* option takes the number of characters to print as its argument. No more than that many characters of the window name are visible.

If you wanted to use the **WindowList** as an icon manager, you could invoke the following:

```
WindowList OnlyIcons, Sticky, OnTop, Geometry
```

(Note − the *Only* options essentially wipe out all other ones... but the *OnlyListSkip* option which just causes **WindowList** to only consider the windows with *WindowListSkip* style.)

XSync

When **XSync** is called, the X function with the same name is used to send all pending X requests to the server. This command is intended for debugging only.

XSynchronize [bool]

The **XSynchronize** command controls whether X requests are sent to the X server immediately or not. Normally, requests are sent in larger batches to save unnecessary communication. To send requests immediately, use "True" as the argument, to disable this use "False" or to toggle between both methods use "Toggle" or omit the **bool** argument. Fvwm defaults to synchronized requests when started with the **−−debug** option. This command is intended for debugging only.

+

Used to continue adding to the last specified decor, function or menu. See the discussion for **AddToDecor**, **AddToFunc**, and **AddToMenu**.

Window Movement and Placement

AnimatedMove *x y* [Warp]

Move a window in an animated fashion. Similar to**Mo ve** command. The options are the same, except they are required, since it doesn't make sense to have a user move the window interactively and animatedly. If the optional argument *Warp* is specified the pointer is warped with the window.

HideGeometryWindow [Never | Move | Resize]

Hides the position or size window that is usually shown when a window is moved or resized interactively. To switch it off only for move or resize operations the optional parameters *Move* and *Resize* can be used respectively. To switch both on again use the *Never* option.

Layer [*arg1 arg2*] | [default]

Puts the current window in a new layer. If*ar g1* is non zero then the next layer is the current layer number plus *arg1*. If*ar g1* is zero then the new layer is *arg2*.

As a special case, *default* puts the window in its default layer, i.e. the layer it was initially in. The same happens if no or invalid arguments are specified.

Lower

Allows the user to lower a window. Note that this lowers a window only in its layer. To bring a window to the absolute bottom, use

```
AddToFunc lower−to−bottom
+ I Layer 0 0
+ I Lower
```

Move [[screen *screen*] [w | m]*x*[p | w] ... [w | m]*y*[p | w] ... [Warp]] | [pointer] | [ewmhiwa]

Allows the user to move a window. If called from somewhere in a window or its border, then that window is moved. If called from the root window then the user is allowed to select the target window. By default, the EWMH working area is honoured.

If the literal option *screen* followed by a *screen* argument is specified, the coordinates are interpreted as relative to the given screen. The width and height of the screen are used for the calculations instead of the display dimensions. The *screen* as interpreted as in the **MoveToScreen** command. If the optional argument *Warp* is specified the pointer is warped with the window. If the single argument *pointer* is given, the top left corner of the window is moved to the pointer position before starting the operation; this is mainly intended for internal use by modules like **FvwmPager**. If the optional argument *ewmhiwa* is given, then the window position will ignore the working area (such as ignoring any values set via **EwmhBaseStruts**).

The operation can be aborted with Escape or any mouse button not set to place the window. By default mouse button 2 is set to cancel the move operation. To change this you may use the **Mouse** command with special context 'P' for Placement.

The window condition *PlacedByButton* can be used to check if a specific button was pressed to place the window (see **Current** command).

If the optional arguments *x* and *y* are provided, then the window is moved immediately without user interaction. Each argument can specify an absolute or relative position from either the left/top or right/bottom of the screen. By default, the numeric value given is interpreted as a percentage of the screen width/height, but a trailing '*p*' changes the interpretation to mean pixels, while a trailing '*w*' means precent of the window width/height. To move the window relative to its current position, add the '*w*' (for "window") prefix before the *x* and/or *y* value. To move the window to a position relative to the current location of the pointer, add the '*m*' (for "mouse") prefix. To leave either coordinate unchanged, "*keep*" can be specified in place of *x* or *y*.

For advanced uses, the arguments *x* and *y* can be used multiple times, but without the prefix '*m*' or '*w*'. (See complex examples below).

Simple Examples:

```
# Interactive move
Mouse 1 T A Move
# Move window to top left is at (10%,10%)
Mouse 2 T A Move 10 10
# Move top left to (10pixels,10pixels)
Mouse 3 T A Move 10p 10p
```

More complex examples (these can be bound as actions to keystrokes, etc.; only the command is shown, though):

```
# Move window so bottom right is at bottom
# right of screen
Move −0 −0

# Move window so top left corner is 10 pixels
# off the top left screen edge
Move +−10 +−10

# Move window 5% to the right, and to the
# middle vertically
Move w+5 50

# Move window up 10 pixels, and so left edge
# is at x=40 pixels
Move 40p w−10p
```

```
# Move window to the mouse pointer location
Move m+0 m+0

# Move window to center of screen (50% of screen
# poition minus 50% of widow size).
Move 50−50w 50−50w
```

Note: In order to obtain moving windows which do not snap to screen, with interactive move, hold down *Alt* whilst moving the window to disable snap attraction if it's defined.

See also the **AnimatedMove** command.

MoveToDesk [prev | *arg1* [*arg2*] [*min max*]]
Moves the selected window to another desktop. The arguments are the same as for the **GotoDesk** command. Without any arguments, the window is moved to the current desk. **MoveToDesk** is a replacement for the obsolete **WindowsDesk** command, which can no longer be used.

MoveThreshold [*pixels*]
When the user presses a mouse button upon an object fvwm waits to see if the action is a click or a drag. If the mouse moves by more than *pixels* pixels it is assumed to be a drag.

Previous versions of fvwm hardwired *pixels* to 3, which is now the default value. If*pixels* is negative or omitted the default value (which might be increased when 16000x9000 pixel displays become affordable) is restored.

MoveToPage [*options*] [*x*[p | w] *y*[p | w]] | [prev]
Moves the selected window to another page (*x*,*y*). The upper left page is (0,0), the upper right is (M,0), where M is one less than the current number of horizontal pages specified in the **DesktopSize** command. Similarly the lower left page is (0,N), and the lower right page is (M,N). Negative page numbers refer to pages from the rightmost/lowest page. If *x* and *y* are not given, the window is moved to the current page (a window that has the focus but is off−screen can be retrieved with this). Moving windows to a page relative to the current page can be achieved by adding a trailing '*p*' after any or both numerical arguments. To move the window relative to its current location, add a trailing '*w*'. To move a window to the previous page use *prev* as the single argument.

Windows are usually not moved beyond desk boundaries.

Possible *options* are *wrapx* and *wrapy* to wrap around the x or y coordinate when the window is moved beyond the border of the desktop. For example, with *wrapx*, when the window moves past the right edge of the desktop, it reappears on the left edge. The options *nodesklimitx* and *nodesklimity* allow moving windows beyond the desk boundaries in x and y direction (disabling the *wrapx* and *wrapy* options).

Examples:

```
# Move window to page (2,3)
MoveToPage 2 3

# Move window to lowest and rightmost page
MoveToPage −1 −1

# Move window to last page visited
MoveToPage prev

# Move window two pages to the right and one
# page up, wrap at desk boundaries
MoveToPage wrapx wrapy +2p −1p
```

MoveToScreen [*screen*]
Moves the selected window to another Xinerama screen. The *screen* argument can be '*p*' for the primary screen, '*c*' for the current screen (containing the mouse pointer), '*w*' for the screen containing the center of +the context window, '*g*' for the global screen or the screen number itself (counting from zero).

OpaqueMoveSize [*percentage*]
Tells fvwm the maximum size window with which opaque window movement should be used. The percentage is percent of the total screen area (may be greater than 100). With

OpaqueMoveSize 0

all windows are moved using the traditional rubber−band outline. With

OpaqueMoveSize *unlimited*

or if a negative percentage is given all windows are moved as solid windows. The default is

OpaqueMoveSize 5

which allows small windows to be moved in an opaque manner but large windows are moved as rubber−bands. If*per centage* is omitted or invalid the default value is set. To resize windows in an opaque manner you can use the *ResizeOpaque* style. See the **Style** command.

PlaceAgain [Anim] [Icon]
Causes the current window's position to be re−computed using the initial window placement logic. The window is moved to where it would have been if it were a new window that had just appeared. Most useful with *Smart* or *Clever* (*ReallySmart*) placement. With the optional argument *Anim* an animated move is used to place the window in its new position. With the additional option *Icon*, the icon is placed again instead.

Raise
Allows the user to raise a window. Note that this raises a window only in its layer. To bring a window to the absolute top, use

AddToFunc raise−to−top
+ I **Layer** 0 ontop
+ I Raise

where ontop is the highest layer used in your setup.

RaiseLower
Alternately raises and lowers a window. The window is raised if it is obscured by any window (except for its own transients when *RaiseTransient* style is used; see the **Style** command) otherwise it is lowered.

Resize [[frame] [direction *dir*] [warptoborder *automatic*] [fixeddirection] [w]*width*[p | c | wa | da] [w]*height*[p | c]]
Allows for resizing a window. If called from somewhere in a window or its border, then that window is resized. If called from the root window then the user is allowed to select the target window.

The operation can be aborted with Escape or by pressing any mouse button (except button 1 which confirms it).

If the optional arguments *width* and *height* are provided, then the window is resized so that its dimensions are *width* by *height*. The units of*width* and *height* are percent−of−screen, unless a letter '*p*' is appended to one or both coordinates, in which case the location is specified in pixels. With a '*c*' suffix the unit defined by the client application (hence the c) is used. With the suffix '*wa*' the value is a percentage of the width or height size of the EWMH working area, and with the suffix '*da*' it is a percentage of the width or height of the EWMH dynamic working area. So you

can say

Resize 80c 24c

to make a terminal window just big enough for 80x24 characters.

If the *width* or *height* is prefixed with the letter '*w*' the size is not taken as an absolute value but added to the current size of the window. Example:

Enlarge window by one line
Resize keep w+1c

Both, *width* and *height* can be negative. In this case the new size is the screen size minus the given value. If either value is "*keep*", the corresponding dimension of the window is left untouched. The new size is the size of the client window, thus

Resize 100 100

may make the window bigger than the screen. To base the new size on the size of the whole fvwm window, add the *frame* option after the command. The options *fixeddirection*, *direction* and *warptoborder* are only used in interactive move operations. With *fixeddirection* the same border is moved even if the pointer moves past the opposite border. The*dir ection* option must be followed by a direction name such as "NorthWest", "South" or "East" (you get the idea). Resizing is started immediately, even if the pointer is not on a border. If the special option*automatic* is gi ven as a direction argument, then the direction to resize is calculated based on the position of the pointer in the window. If the pointer is in the middle of the window, then no direction is calculated. The *warptoborder* option can be used to warp the pointer to the direction indicated. As with the *automatic* option for *direction*, the border to warp to is calculated based on the pointer's proximity to a given border. Also, if resizing is started by clicking on the window border, the pointer is warped to the outer edge of the border.

AddToFunc ResizeSE I Resize **Direction** SE
Mouse 3 A M ResizeSE

Resize [bottomright | br *x y*]

An alternate syntax is used if the keyword *bottomright* or in short *br* follows the command name. In this case, the arguments *x* and *y* specify the desired position of the bottom right corner of the window. They are interpreted exactly like the *x* and *y* arguments of the **Move** command. Actually, any of the options accepted by the **Move** command can be used.

ResizeMaximize [*resize−arguments*]

Combines the effects of **Resize** and **Maximize** in a single command. When used on a maximized window, the window is resized and is still in the maximized state afterwards. When used on an unmaximized window, the window is resized and put into the maximized state afterwards. This is useful if the user wants to resize the window temporarily and then return to the original geometry. The *resize−arguments* are the same as for the **Resize** command.

ResizeMove *resize−arguments move−arguments*

This command does the same as the **Resize** and **Move** commands, but in a single call which is less visually disturbing. The *resize−arguments* are exactly the same arguments as for the **Resize** command and the *move−arguments* are exactly the same arguments as for the **Move** command except the *pointer* option which is not supported by the **ResizeMove** command.

Examples:

Move window to top left corner and cover
most of the screen
ResizeMove −10p −20p 0 0

```
# Grow the focused window towards the top of screen
Current Resize keep w+$[w.y]p keep 0
```

Note: Fvwm may not be able to parse the command properly if the option *bottomright* of the **Resize** command is used.

ResizeMoveMaximize *resize−arguments move−arguments*
Combines the effects of **ResizeMove** and **Maximize** in a single command. When used on a maximized window, the window is resized and moved and is still in the maximized state afterwards. When used on an unmaximized window, the window is resized and put into the maximized state afterwards. This is useful if the user wants to resize the window temporarily and then return to the original geometry. Ther*esize−arguments* and *move−argument*s are the same as for the **ResizeMove** command.

RestackTransients
This command regroups the transients of a window close to it in the stacking order as if the window had just been lowered and then raised. The position of the window itself is not altered. Only windows that use either the *RaiseTransient* or *LowerTransient* style are affected at all. When **RestackTransients** is used on a transient window with the *StackTransientParent* style set, it is redirected to the parent window.

SetAnimation *milliseconds−delay* [*fractions−to−move−list*]
Sets the time between frames and the list of fractional offsets to customize the animated moves of the **AnimatedMove** command and the animation of menus (if the menu style is set to animated; see **MenuStyle** command). If the *fractions−to−move−list* is omitted, only the time between frames is altered. The *fractions−to−move−list* specifies how far the window should be offset at each successive frame as a fraction of the difference between the starting location and the ending location. e.g.:

```
SetAnimation 10 −.01 0 .01 .03 .08 .18 .3 \
  .45 .6 .75 .85 .90 .94 .97 .99 1.0
```

Sets the delay between frames to 10 milliseconds, and sets the positions of the 16 frames of the animation motion. Negative values are allowed, and in particular can be used to make the motion appear more cartoonish, by briefly moving slightly in the opposite direction of the main motion. The above settings are the default.

SnapAttraction [*proximity* [*behaviour*] [Screen]]
The **SnapAttraction** command is obsolete. It has been replaced by the **Style** command option *SnapAttraction*.

SnapGrid [*x−grid−size y−grid−size*]
The **SnapGrid** command is obsolete. It has been replaced by the **Style** command option *SnapGrid*.

WindowsDesk *arg1* [*arg2*]
Moves the selected window to another desktop.

This command has been removed and must be replaced by **MoveToDesk**, the arguments for which are the same as for the **GotoDesk** command.

Important
You cannot simply change the name of the command: the syntax has changed. If you used:

```
WindowsDesk n
```

to move a window to desk n, you have to change it to:

```
MoveToDesk 0 n
```

XorPixmap [*pixmap*]
Selects the pixmap with which bits are xor'ed when doing rubber−band window moving or resizing. This has a better chance of making the rubber−band visible if**XorV alue** does not give good results. An example pixmap *resize.rainbow.xpm* is provided with the icon distribution. To turn the *XorPixmap* off again use the **XorValue** command or omit the *pixmap* argument.

XorValue [*number*]
Changes the value with which bits are xor'ed when doing rubber−band window moving or resizing. Valid values range from zero to the maximum value of an unsigned long integer on your system. Setting this value is a trial−and−error process. The default value 0 tries to find a value that gives a good contrast to black and white. The default value is used if the given *number* is omitted or invalid.

Focus & Mouse Movement

CursorMove *horizontal*[p] *vertical*[p]
Moves the mouse pointer by *horizontal* pages in the X direction and *vertical* pages in the Y direction. Either or both entries may be negative. Both horizontal and vertical values are expressed in percent of pages, so

CursorMove 100 100

means to move down and right by one full page.

CursorMove 50 25

means to move right half a page and down a quarter of a page. Alternatively, the distance can be specified in pixels by appending a '*p*' to the horizontal and/or vertical specification. For example

CursorMove −10p −10p

means move ten pixels up and ten pixels left. The CursorMove function should not be called from pop−up menus.

FlipFocus [NoWarp]
Executes a **Focus** command as if the user had used the pointer to select the window. This command alters the order of the WindowList in the same way as clicking in a window to focus, i.e. the target window is removed from the **WindowList** and placed at the start. This command is recommended for use with the **Direction** command and in the function invoked from **WindowList**.

Focus [NoWarp]
Sets the keyboard focus to the selected window. If the*NoW arp* argument is given, this is all it does. Otherwise it also moves the viewport or window as needed to make the selected window visible. This command does not automatically raise the window. Does not warp the pointer into the selected window (see **WarpToWindow** function). Does not de−iconify. This command does not alter the order of the **WindowList**, it rotates the **WindowList** around so that the target window is at the start.

When the *NoWarp* argument is given, Focus cannot transfer the keyboard focus to windows on other desks.

To raise and/or warp a pointer to a window together with **Focus** or **FlipFocus**, use a function, like:

AddToFunc SelectWindow
+ I Focus
+ I **Iconify** false
+ I **Raise**
+ I **WarpToWindow** 50 8p

WarpToWindow [!raise | raise] *x*[p] *y*[p]
Warps the cursor to the associated window and raises it (unless the option *!raise* is present). The parameters *x* and *y* default to percentage of window down and in from the upper left hand corner (or number of pixels down and in if *'p'* is appended to the numbers). If a number is negative the opposite edge is used and the direction reversed. This command works also with windows that are not managed by fvwm. In this case fvwm does not bring the window onto the screen if it is not visible. For example it is possible to warp the pointer to the center of the root window on screen 1:

WindowId root 1 WarpToWindow 50 50

Window State

Close
If the window accepts the delete window protocol a message is sent to the window asking it to gracefully remove itself. If the window does not understand the delete window protocol then the window is destroyed as with the **Destroy** command. Note: if the window accepts the delete window protocol but does not close itself in response, the window is not deleted.

Delete
Sends a message to a window asking that it remove itself, frequently causing the application to exit.

Destroy
Destroys an application window, which usually causes the application to crash and burn.

Iconify [*bool*]
Iconifies a window if it is not already iconified or de−iconifies it if it is already iconified. The optional argument *bool* is a boolean argument. "*True*" means only iconification is allowed, while "*False*" forces de−iconification. Using "*toggle*" switches between iconified and de−iconified states.

There are a number of **Style** options which influence the appearance and behavior of icons (e.g. *StickyIcon*, *NoIcon*).

For backward compatibility, the optional argument may also be a positive number instead of "True", or a negative number instead of "False". Note that this syntax is obsolete, and will be removed in the future.

Maximize [*flags*] [*bool* | *forget*] [*horizontal*[p]] [*vertical*[p]]
Without its optional arguments (or if the *bool* bit has the value "*toggle*") **Maximize** causes the window to alternately switch from a full−screen size to its normal size. To force a window into maximized (normal) state you can use a "*True*" or "*False*" value for the *bool* argument.

With just the parameter "forget" a maximized window reverts back into normal state but keeps its current maximized size. This can be useful in conjunction with the commands **ResizeMaximize** and **ResizeMoveMaximize**. If the window is not maximized, nothing happens.

With the optional arguments *horizontal* and *vertical*, which are expressed as percentage of a full screen, the user can control the new size of the window. An optional suffix *'p'* can be used to indicate pixels instead of percents of the screen size. If horizontal is greater than 0 then the horizontal dimension of the window is set to *horizontal**screen_width/100. If the value is smaller than 0 the size is subtracted from the screen width, i.e. −25 is the same as 75. If *horizontal* is "*grow*", it is maximized to curren available space until finding any obstacle. The vertical resizing is similar. If both horizontal and vertical values are "grow", it expands vertically first, then horizontally to find space. Instead of the horizontal "grow" argument, "*growleft*" or "*growright*" can be used respectively "*growup*" and "*growdown*". The optional*fla gs* argument is a space separated list containing the following key words: *fullscreen*, *ewmhiwa*, *growonwindowlayer*, *growonlayers* and *screen*. *fullscreen* causes the window to become fullscreened if the appropriate EWMH hint is set. *ewmhiwa* causes fvwm to ignore the EWMH working area. *growonwindowlayer* causes the various grow methods to ignore windows with a layer other than

the current layer of the window which is maximized. The *growonlayers* option must have two integer arguments. The first one is the minimum layer and the second one the maximum layer to use. Windows that are outside of this range of layers are ignored by the grow methods. A negative value as the first or second argument means to assume no minimum or maximum layer. *screen* must have an argument which specifies the Xinerama screen on which to operate. It can be 'p' for the primary screen, 'c' for the current screen (containing the mouse pointer), 'g' for the global screen or the screen number itself (counting from zero). This option is only useful with multiple Xinerama screens.

Here are some examples. The following adds a title−bar button to switch a window to the full vertical size of the screen:

Mouse 0 4 A Maximize 0 100

The following causes windows to be stretched to the full width:

Mouse 0 4 A Maximize 100 0

This makes a window that is half the screen size in each direction:

Mouse 0 4 A Maximize 50 50

To expand a window horizontally until any other window is found:

Mouse 0 4 A Maximize 0 grow

To expand a window until any other window on the same or a higher layer is hit.

Mouse 0 4 A Maximize growonlayers $[w.layer] −1 grow grow

To expand a window but leave the lower 60 pixels of the screen unoccupied:

Mouse 0 4 A Maximize 100 −60p

Values larger than 100 can be used with caution.

Recapture

This command is obsolete and should not be used anymore. Should you want to do something specific that you cannot do without it, please report this to the fvwm−workers mailing list <fvwm−workers@fvwm.org>. This command may be removed at some point in the future. Please read the note at the end of the section **Delayed Execution of Commands** to learn about how to avoid the **Recapture** command.

Causes fvwm to recapture all of its windows. This ensures that the latest style parameters are used. The recapture operation is visually disturbing.

Since fvwm version 2.4 only a very few **Style** options need a **Recapture** to take effect (e.g. *UseStyle*).

RecaptureWindow

This command is obsolete and should not be used anymore. See**Recaptur e** For details.

Causes fvwm to recapture the chosen window.

Refresh

Causes all windows on the screen to redraw themselves. All pending updates of all windows' styles and looks are applied immediately. E.g. if**Style** or **T itleStyle** commands were issued inside a fvwm function.

RefreshWindow

Causes the chosen window to redraw itself. All pending updates of the window's style and look are applied immediately. E.g. if**Style** or **T itleStyle** commands were issued inside a fvwm function.

Stick [*bool*]
If the *bool* argument is empty or "*toggle*", the **Stick** command makes a window sticky if it is not already sticky, or non−sticky if it is already sticky. To make a window sticky regardless of its current state the *bool* argument must be "True". To make it non−sticky use "False".

StickAcrossPages [*bool*]
Works like **Stick** but only sticks a window across pages, not across desks.

StickAcrossDesks [*bool*]
Works like **Stick** but only sticks a window across desks, not across pages.

WindowShade [bool] | [[ShadeAgain] *direction*]
Toggles the window shade feature for titled windows. Windows in the shaded state only display a title−bar. If*bool* is not gi ven or "*toggle*", the window shade state is toggled. If *bool* is "True", the window is forced to the shaded state. If *bool* is "False", then the window is forced to the non−shaded state. To force shading in a certain direction, the *direction* argument can be used. Any of the strings "*North*", "*South*", "*West*", "*East*", "*NorthWest*", "*NorthEast*", "*SouthWest*", "*SouthEast*" or "*Last*" can be given. The direction can be abbreviated with the usual one or two letters "*N*", "*NW*", etc. Using a direction on a window that was already shaded unshades the window. To shade it in a different direction, use the *ShadeAgain* option. The direction *Last* shades the window in the direction it last was shaded. If the window has never been shaded before it is shaded as if no direction had been given. Windows without titles can be shaded too. Please refer also to the options *WindowShadeSteps*, *WindowShadeShrinks*, *WindowShadeScrolls*, *WindowShadeLazy*, *WindowShadeAlwaysLazy* and *WindowShadeBusy* options of the **Style** command. Examples:

```
Style * WindowShadeShrinks, WindowShadeSteps 20, \
    WindowShadeLazy
Mouse 1 − S WindowShade North
Mouse 1 [ S WindowShade West
Mouse 1 ] S WindowShade E
Mouse 1 _ S WindowShade S
```

Note: When a window that has been shaded with a *direction* argument changes the direction of the window title (see *TitleAtTop* **Style** option), the shading direction does not change. This may look very strange. Windows that were shaded without a *direction* argument stay shaded in the direction of the title bar.

For backward compatibility, the optional argument may also be 1 to signify "on", and 2 to signify "off". Note that this syntax is obsolete, and will be removed in the future.

WindowShadeAnimate [*steps* [p]]
This command is obsolete. Please use the *WindowShadeSteps* option of the **Style** command instead.

Mouse, Key & Stroke Bindings

IgnoreModifiers [*Modifiers*]
Tells fvwm which modifiers to ignore when matching Mouse or Key bindings. **IgnoreModifiers** affects the *ClickToFocus* style too. This command belongs into your *config*. If you issue it when your fvwm session is already up and running the results are unpredictable. The should appear before any applications or modules are started in your *config* file (e.g. with the **Exec** command).

Modifiers has the same syntax as in the **Mouse** or **Key** bindings, with the addition of 'L' meaning the caps lock key. The default is "L". *Modifiers* can be omitted, meaning no modifiers are ignored. This command comes in handy if the num-lock and scroll-lock keys interfere with your shortcuts. With XFree86 '2' usually is the num-lock modifier and '5' refers to the scroll-lock key. To turn all these pesky modifiers off you can use this command:

```
IgnoreModifiers L25
```

If the *Modifiers* argument is the string "*default*", fvwm reverts back to the default value "L".

Important
This command creates a lot of extra network traffic, depending on your CPU, network connection, the number of **Key** or **Mouse** commands in your configuration file and the number of modifiers you want to ignore. If you do not have a lightning fast machine or very few bindings you should not ignore more than two modifiers. I.e. do not ignore scroll-lock if you have no problem with it. In the *FAQ* you can find a better solution of this problem.

EdgeCommand [*direction* [*Function*]]
Binds a specified fvwm command *Function* to an edge of the screen. Direction may be one of "*North*", "*Top*", "*West*", "*Left*", "*South*", "*Bottom*", "*Right*" and "*East*". If*Function* is omitted the binding for this edge is removed. If EdgeCommand is called without any arguments all edge bindings are removed.

Function is executed when the mouse pointer enters the invisible pan frames that surround the visible screen. The binding works only if **EdgeThickness** is set to a value greater than 0. If a function is bound to an edge, scrolling specified by **EdgeScroll** is disabled for this edge. It is possible to bind a function only to some edges and use the other edges for scrolling. This command is intended to raise or lower certain windows when the mouse pointer enters an edge. **FvwmAuto** can be used get a delay when raising or lowering windows. The following example raises **FvwmButtons** if the mouse pointer enters the top edge of the screen.

```
# Disable EdgeScrolling but make it possible
# to move windows over the screen edge
EdgeResistance -1
Style * EdgeMoveDelay 250
Style * EdgeMoveResistance 20

# Set thickness of the edge of the screen to 1
EdgeThickness 1

# Give focus to FvwmButtons if the mouse
# hits top edge
EdgeCommand Top Next (FvwmButtons) Focus
# Make sure the Next command matches the window
Style FvwmButtons CirculateHit

Module FvwmButtons
Module FvwmAuto 100 "Silent AutoRaiseFunction" \
    "Silent AutoLowerFunction"

# If any window except FvwmButtons has
# focus when calling this function
# FvwmButtons are lowered
DestroyFunc AutoLowerFunction
AddToFunc AutoLowerFunction
+ I Current (!FvwmButtons) All (FvwmButtons) Lower

# If FvwmButtons has focus when calling this function raise it
DestroyFunc AutoRaiseFunction
AddToFunc AutoRaiseFunction
+ I Current (FvwmButtons) Raise
```

Normally, the invisible pan frames are only on the screen edges that border virtual pages. If a screen edge has a command bound to it, the pan frame is always created on that edge.

EdgeLeaveCommand [*direction* [*Function*]]

Binds a specified fvwm command *Function* to an edge of the screen. Direction may be one of "*North*", "*Top*", "*West*", "*Left*", "*South*", "*Bottom*", "*Right*" and "*East*". If*Function* is omitted the binding for this edge is removed. If EdgeLeaveCommand is called without any arguments all edge bindings are removed.

Function is executed when the mouse pointer leaves the invisible pan frames that surround the visible screen. The binding works only if **EdgeThickness** is set to a value greater than 0. If a function is bound to an edge, scrolling specified by **EdgeScroll** is disabled for this edge. It is possible to bind a function only to some edges and use the other edges for scrolling. This command is intended to raise or lower certain windows when the mouse pointer leaves an edge. **FvwmAuto** can be used get a delay when raising or lowering windows. See example for **EdgeCommand**

Normally, the invisible pan frames are only on the screen edges that border virtual pages. If a screen edge has a command bound to it, the pan frame is always created on that edge.

Key [(*window*)] *Keyname Context Modifiers Function*

Binds a keyboard key to a specified fvwm command, or removes the binding if *Function* is '−'. The syntax is the same as for a **Mouse** binding except that the mouse button number is replaced with a *Keyname*. Normally, the key binding is activated when the key is pressed. *Keyname* is a standard X11 key name as defined in */usr/include/X11/keysymdef.h*, (without the *XK_* prefix), or the keysym database */usr/X11R6/lib/X11/XKeysymDB*. Only key names that are generated with no modifier keys or with just the Shift key held are guaranteed to work. The*Conte xt* and *Modifiers* fields are defined as in the **Mouse** binding. However, when you press a key the context window is the window that has the keyboard focus. That is not necessarily the same as the window the pointer is over (with *SloppyFocus* or *ClickToFocus*). Note that key bindings with the '*R*' (root window) context do not work properly with *SloppyFocus* and *ClickToFocus*. If you encounter problems, use the **PointerKey** command instead. If you want to bind keys to a window with *SloppyFocus* or *ClickToFocus* that are supposed to work when the pointer is not over the window, fvwm assumes the pointer is over the client window (i.e. you have to use the 'W' context).

The special context '*M*' for menus can be used to (re)define the menu controls. It be used alone or together with 'T', 'S', 'I', '[', ']', '−' and '_'. See the **Menu Bindings** section for details.

The following example binds the built−in window list to pop up when Alt-Ctrl-Shift-F11 is hit, no matter where the mouse pointer is:

```
Key F11 A SCM WindowList
```

Binding a key to a title−bar button causes that button to appear. Please refer to the**Mouse** command for details.

Mouse [(*window*)] *Button Context Modifiers Function*

Defines a mouse binding, or removes the binding if *Function* is '−'. *Button* is the mouse button number. If*Button* is zero then an y button performs the specified function. Note that only mouse buttons 1 to 5 are fully supported by X11. Any number above this works only partially. Complex functions can not be used with these buttons and neither any operation that requires dragging the pointer with the button held. This is due to limitations of X11. By default, the highest allowed button number is 9.

Context describes where the binding applies. Valid contexts are '*R*' for the root window, '*W*' for an application window, '*D*' for a desktop application (as kdesktop or Nautilus desktop), '*T*' for a window title−bar, '*S*' for a window side, top, or bottom bar, '*[*', '*]*', '-' and '_' for the left, right, top or bottom side only, '*F*' for a window frame (the corners), '<', '^', '>' and '*v*' for the top left, top right, bottom right or bottom left corner, '*I*' for an icon window, or '*0*' through '*9*' for title−bar buttons, or any combination of these letters. '*A*' is for any context. For instance, a context of "FST" applies when the mouse is anywhere in a window's border except the title−bar buttons. Only 'S' and 'W' are valid for an undecorated window.

The special context '*M*' for menus can be used to (re)define the menu controls. It can be used alone or together with 'T', 'S', 'I', '[', ']', '−' and '_'. See the **Menu Bindings** section for details.

The special context '*P*' controls what buttons that can be used to place a window. When using this context no modifiers are allowed (*Modifiers* must be N), no *window* is allowed, and the *Function* must be one of *PlaceWindow*, *PlaceWindowDrag*, *PlaceWindowInteractive*, *CancelPlacement*, *CancelPlacementDrag*, *CancelPlacementInteractive* or −.

PlaceWindow makes *Button* usable for window placement, both for interactive and drag move. *CancelPlacement* does the inverse. That is makes *Button* to cancel move for both interactive and drag move. It may however not override how new windows are resized after being placed. This is controlled by the **Emulate** command. Also a window being dragged can always be placed by releasing the button hold while dragging, regardless of if it is set to *PlaceWindow* or not.

PlaceWindowDrag and *PlaceWindowInteractive*/*CancelPlacementDrag* and *CancelPlacementInteractive* work as *PlaceWindow*/*CancelPlacement* with the exception that they only affect either windows dragged / placed interactively.

- is equivalent to *CancelPlacement*.

The following example makes all buttons but button 3 usable for interactive placement and makes drag moves started by other buttons than one cancel if button 1 is pressed before finishing the move:

```
Mouse 0 P N PlaceWindow
Mouse 3 P N CancelPlacement
Mouse 1 P N CancelPlacementDrag
```

By default, the binding applies to all windows. You can specify that a binding only applies to specific windows by specifying the window name in brackets. The window name is a wildcard pattern specifying the class, resource or name of the window you want the binding to apply to.

The following example shows how the same key−binding can be used to perform different functions depending on the window that is focused:

```
Key (rxvt)  V A C Echo ctrl−V−in−RXVT
Key (*term) V A C Echo ctrl−V−in−Term
Key (*vim)  V A C −−
Key         V A C Echo ctrl−V−elsewhere
```

A '--' action indicates that the event should be propagated to the specified window to handle. This is only a valid action for window−specific bindings.

This example shows how to display the WindowList when Button 3 is pressed on an rxvt window:

```
Mouse (rxvt) 3 A A WindowList
```

Note that Fvwm actually intercepts all events for a window−specific binding and (if the focused window doesn't match any of the bindings) sends a synthetic copy of the event to the window. This should be transparent to most applications, however (for security reasons) some programs ignore these synthetic events by default − xterm is one of them. To enable handling of these events, add the following line to your *~/.Xdefaults* file:

```
XTerm*allowSendEvents:  true
```

Modifiers is any combination of '*N*' for no modifiers, '*C*' for control, '*S*' for shift, '*M*' for Meta, '*L*' for Caps−Lock or '*A*' for any modifier. For example, a modifier of "SM" applies when both the Meta and Shift keys are down. X11 modifiers mod1 through mod5 are represented as the digits '1' through '5'. The modifier 'L' is ignored by default. To turn it on, use the **IgnoreModifiers** command.

Function is one of fvwm's commands.

The title−bar buttons are numbered with odd numbered buttons on the left side of the title−bar and even numbers on the right. Smaller−numbered buttons are displayed toward the outside of the window while larger−numbered buttons appear toward the middle of the window (0 is short for 10). In summary, the buttons are numbered:

1 3 5 7 9 0 8 6 4 2

The highest odd numbered button which has an action bound to it determines the number of buttons drawn on the left side of the title bar. The highest even number determines the number of right side buttons which are drawn. Actions can be bound to either mouse buttons or keyboard keys.

PointerKey [(*window*)] *Keyname Context Modifiers Function*

This command works exactly like the **Key** command. The only difference is that the binding operates on the window under the pointer. Normal key bindings operate on the focused window instead. The**P ointerKey** command can for example be used to bind keys to the root window if you are using *SloppyFocus* or *ClickToFocus*. However, some applications (xterm is one example) are unable to handle this key anymore, even if the pointer is over the xterm window. It is recommended to use the **PointerKey** command only for key combinations that are not needed in any application window.

Example:

Style * *SloppyFocus*
PointerKey f1 a m **Menu** MainMenu

Stroke [(*window*)] *Sequence Button Context Modifiers Function*

Binds a mouse stroke sequence to a specified fvwm command, or removes the binding if *Function* is '−'. The syntax is the same as for a **Mouse** binding except that *Sequence* is inserted in front of the button number and a value of 0 for *Button* concerns the **StrokeFunc** command. The *Context* and *Modifiers* fields are defined as in the **Mouse** binding. However, only the '*R*' Context really works (if you want to use other contexts you need to use the **StrokeFunc** below).

Strokes sequences are defined in a telephone grid like this:

1 2 3

4 5 6

7 8 9

or in a numeric pad grid like this:

7 8 9

4 5 6

1 2 3

The telephone grid is used by default, to use the numeric pad grid you should begin the sequence with a '*N*'. Note that a complex motion may produce several different sequences (see the "netscape" example below to handle such motion). Moreover, sequences are limited to 20 elements (with the present version of **libstroke**), however, in practice it is preferable to use sequence with less than 12 elements.

Because of the default button menu in fvwm, you may need to remove a mouse button binding (using an empty action) before using the stroke

Mouse 3 R N

Also, you can still use the stroke "sequence 0" to simulate a click:

Stroke 0 3 R N **Menu WindowList Nop**

The following example starts xterm when the mouse drags an 'I' on the root window with button 3 pressed down:

Stroke 258 3 R N **Exec** exec xterm

An example for Netscape:

Stroke 7415963 3 R N **Exec** exec netscape
Stroke 74148963 3 R N **Exec** exec netscape
Stroke 74158963 3 R N **Exec** exec netscape
Stroke 7418963 3 R N **Exec** exec netscape
Stroke 415963 3 R N **Exec** exec netscape

You may prefer to use the numeric pad grid since you have such a grid on your machine. Here an example:

Stroke N78963214 3 R N **FvwmForm** FvwmForm−QuitVerify
Stroke N789632147 3 R N **FvwmForm** FvwmForm−QuitVerify

This example starts the "QuitVerify" form if you draw a box that begins in the top left corner.

Note: You need **libstroke** installed and fvwm compiled with stroke support. **libstroke** can be obtained at *http://www.etla.net/~willey/projects/libstroke/*

StrokeFunc [*Options*]

Causes fvwm to record a mouse stroke sequence and to execute the corresponding action as defined in a **Stroke** command. The cursor is modified to the *STROKE* context of the **CursorStyle** command during recording. When the stroke is finished **StrokeFunc** looks for a stroke binding of the form

Stroke sequence 0 Context Modifiers action

and executes the corresponding action (Note the 0). Normal use of this function is via a **Mouse** or **Key** command. Examples:

Mouse 3 A M StrokeFunc
Key x R N StrokeFunc

If you press mouse button 3 and Alt anywhere (respectively, press the key x when the cursor is on the root window), then fvwm records the mouse motions until the mouse button 3 (respectively, the x key) is released and then check if the recorded *sequence* corresponds to a stroke binding of the form

"Stroke *sequence* 0 A M *action*"
"Stroke *sequence* 0 R N *action*"

Note that the *Context* and *Modifiers* are taken at the beginning of the execution of the **StrokeFunc** command (so you can release the modifiers before the end of the stroke recording in the case of a mouse binding and if you used, say, a title−bar context the mouse motion can go through an application window). The keys Escape and Delete allow you to abort the command.

The **StrokeFunc** command has five options: *NotStayPressed*, *EchoSequence*, *DrawMotion*, *FeedBack* and *StrokeWidth*. These options are disabled by default. *EchoSequence* causes fvwm to Echo the recorded stroke sequence. *DrawMotion* causes fvwm to draw the mouse motion on the screen. *FeedBack* causes fvwm to display during a fraction of second the cursor of the *WAIT* context of the **CursorStyle** command if the recorded stroke sequence corresponds to a stroke

binding. *StrokeWidth* takes an integer argument, which must be >= 0 and <= 100 and which defines the width of the line for the *DrawMotion* option.

NotStayPressed works only if **StrokeFunc** is used via a **Mouse** or a **Key** command. This option removes the need to have a button or the key pressed during the stroke, but you have to do a mouse click or press the Return or Space key to finish the mouse motion recording (these keys also work without the *NotStayPressed* option).

You can use the **StrokeFunc** "alone". In this case it works as above with the *NotStayPressed* option enabled. However, *Modifiers*, in general, may not work as expected (i.e., in this case use 'A' or 'N' as *Modifiers* in the stroke bindings).

Note that some computers do not support key release events. If that is the case the**Str okeFunc** used via a **Key** command works as if the *NotStayPressed* option is enabled.

Controlling Window Styles

For readability, the commands in this section are not sorted alphabetically. The description of the**Style** command can be found at the end of this section.

FocusStyle *stylename options*

works exactly like the **Style** command, but accepts only the focus policy related styles beginning with "FP". The prefix can be removed, but at the cost of a little bit of time. **FocusStyle** is meant to make the configuration file more readable. Example:

```
FocusStyle * EnterToFocus, !LeaveToUnfocus
```

is equivalent to

```
Style * FPEnterToFocus, !FPLeaveToUnfocus
```

DestroyStyle *style*

deletes the style named *style*. The changes take effect immediately. Note that*style* is not a wild−carded search string, but rather a case−sensitive string that should exactly match the original **Style** command.

Destroying style "*" can be done, but isn't really to be recommended. For example:

```
DestroyStyle Application*
```

This removes all settings for the style named "Application*", NOT all styles starting with "Application".

DestroyWindowStyle

deletes the styles set by the **WindowStyle** command on the selected window. The changes take effect immediately.

UpdateStyles

All pending updates of all windows' styles and looks are applied immediately. E.g. if**Style**, *WindowStyle* or *TitleStyle* commands were issued inside a fvwm function.

Style *stylename options ...*

The **Style** command is used to set attributes of a window to values other than the default or to set the window manager default styles.

stylename can be a window's name, class, visible name, or resource string. It may contain the wildcards '*' and '?', which are matched in the usual Unix filename manner. Multiple style options in a single **Style** command are read from left to right as if they were issued one after each other in separate commands. A given style always overrides all conflicting styles that have been issued earlier (or further left on the same style line).

Note: windows that have no name (WM_NAME) are given a name of "Untitled", and windows that do not have a class (WM_CLASS, res_class) are given class "NoClass" and those that do not have a resource (WM_CLASS, res_name) are given resource "NoResource".

If a window has the resource "fvwmstyle" set, the value of that resource is used in addition to any window names when selecting the style.

options is a comma separated list containing one or more of the following keywords. Each group of style names is separated by slashes ('/'). The last style in these groups is the default. *BorderWidth*, *HandleWidth*, *!Icon* / *Icon*, *MiniIcon*, *IconBox*, *IconGrid*, *IconFill*, *IconSize*, *!Title* / *Title*, *TitleAtBottom* / *TitleAtLeft* / *TitleAtRight* / *TitleAtTop*, *LeftTitleRotatedCW* / *LeftTitleRotatedCCW*, *RightTitleRotatedCCW* / *RightTitleRotatedCW*, *TopTitleRotated* / *TopTitleNotRotated*, *BottomTitleRotated* / *BottomTitleNotRotated*, *!UseTitleDecorRotation* / *UseTitleDecorRotation*, *StippledTitle* / *!StippledTitle*, *StippledIconTitle* / *!StippledIconTitle*, *IndexedWindowName* / *ExactWindowName*, *IndexedIconName* / *ExactIconName*, *TitleFormat* / *IconTitleFormat* / *!Borders* / *Borders*, *!Handles* / *Handles*, *WindowListSkip* / *WindowListHit*, *CirculateSkip* / *CirculateHit*, *CirculateSkipShaded* / *CirculateHitShaded*, *CirculateSkipIcon* / *CirculateHitIcon*, *Layer*, *StaysOnTop* / *StaysOnBottom* / *StaysPut*, *Sticky* / *Slippery*, *StickyAcrossPages* / *!StickyAcrossPages*, *StickyAcrossDesks* / *!StickyAcrossDesks*, *!StickyStippledTitle* / *StickyStippledTitle*, *!StickyStippledIconTitle* / *StickyStippledIconTitle*, *StartIconic* / *StartNormal*, *Color*, *ForeColor*, *BackColor*, *Colorset*, *HilightFore*, *HilightBack*, *HilightColorset*, *BorderColorset*, *HilightBorderColorset*, *IconTitleColorset*, *HilightIconTitleColorset*, *IconBackgroundColorset*, *IconTitleRelief*, *IconBackgroundRelief*, *IconBackgroundPadding*, *Font*, *IconFont*, *StartsOnDesk* / *StartsOnPage* / *StartsAnyWhere*, *StartsOnScreen*, *StartShaded* / *!StartShaded*, *ManualPlacementHonorsStartsOnPage* / *ManualPlacementIgnoresStartsOnPage*, *CaptureHonorsStartsOnPage* / *CaptureIgnoresStartsOnPage*, *RecaptureHonorsStartsOnPage* / *RecaptureIgnoresStartsOnPage*, *StartsOnPageIncludesTransients* / *StartsOnPageIgnoresTransients*, *IconTitle* / *!IconTitle*, *MwmButtons* / *FvwmButtons*, *MwmBorder* / *FvwmBorder*, *MwmDecor* / *!MwmDecor*, *MwmFunctions* / *!MwmFunctions*, *HintOverride* / *!HintOverride*, *!Button* / *Button*, *ResizeHintOverride* / *!ResizeHintOverride*, *OLDecor* / *!OLDecor*, *GNOMEUseHints* / *GNOMEIgnoreHints*, *StickyIcon* / *SlipperyIcon*, *StickyAcrossPagesIcon* / *!StickyAcrossPagesIcon*, *StickyAcrossDesksIcon* / *!StickyAcrossDesksIcon*, *ManualPlacement* / *CascadePlacement* / *MinOverlapPlacement* / *MinOverlapPercentPlacement* / *TileManualPlacement* / *TileCascadePlacement* / *PositionPlacement*, *MinOverlapPlacementPenalties*, *MinOverlapPercentPlacementPenalties*, *DecorateTransient* / *NakedTransient*, *DontRaiseTransient* / *RaiseTransient*, *DontLowerTransient* / *LowerTransient*, *DontStackTransientParent* / *StackTransientParent*, *SkipMapping* / *ShowMapping*, *ScatterWindowGroups* / *KeepWindowGroupsOnDesk*, *UseDecor*, *UseStyle*, *!UsePPosition* / *NoPPosition* / *UsePPosition*, *!UseUSPosition*, *NoUSPosition* / *UseUSPosition*, *!UseTransientPPosition*, *NoTransientPPosition* / *UseTransientPPosition*, *!UseTransientUSPosition* / *NoTransientUSPosition* / *UseTransientUSPosition*, *!UseIconPosition* / *NoIconPosition* / *UseIconPosition*, *Lenience* / *!Lenience*, *ClickToFocus* / *SloppyFocus* / *MouseFocus|FocusFollowsMouse* / *NeverFocus*, *ClickToFocusPassesClickOff* / *ClickToFocusPassesClick*, *ClickToFocusRaisesOff* / *ClickToFocusRaises*, *MouseFocusClickRaises* / *MouseFocusClickRaisesOff*, *GrabFocus* / *GrabFocusOff*, *GrabFocusTransientOff* / *GrabFocusTransient*, *FPFocusClickButtons*, *FPFocusClickModifiers*, *!FPSortWindowlistByFocus* / *FPSortWindowlistByFocus*, *FPClickRaisesFocused* / *!FPClickRaisesFocused*, *FPClickDecorRaisesFocused* / *!FPClickDecorRaisesFocused*, *FPClickIconRaisesFocused* / *!FPClickIconRaisesFocused*, *!FPClickRaisesUnfocused* / *FPClickRaisesUnfocused*, *FPClickDecorRaisesUnfocused* / *!FPClickDecorRaisesUnfocused*, *FPClickIconRaisesUnfocused* / *!FPClickIconRaisesUnfocused*, *FPClickToFocus* / *!FPClickToFocus*, *FPClickDecorToFocus* / *!FPClickDecorToFocus*, *FPClickIconToFocus* / *!FPClickIconToFocus*, *!FPEnterToFocus* / *FPEnterToFocus*, *!FPLeaveToUnfocus* / *FPLeaveToUnfocus*, *!FPFocusByProgram* / *FPFocusByProgram*, *!FPFocusByFunction* / *FPFocusByFunction*, *FPFocusByFunctionWarpPointer* / *!FPFocusByFunctionWarpPointer*, *FPLenient* / *!FPLenient*, *!FPPassFocusClick* / *FPPassFocusClick*, *!FPPassRaiseClick* / *FPPassRaiseClick*, *FPIgnoreFocusClickMotion* / *!FPIgnoreFocusClickMotion*, *FPIgnoreRaiseClickMotion* / *!FPIgnoreRaiseClickMotion*, *!FPAllowFocusClickFunction* / *FPAllowFocusClickFunction*, *!FPAllowRaiseClickFunction* /

FPAllowRaiseClickFunction, *FPGrabFocus* / *!FPGrabFocus*, *!FPGrabFocusTransient* / *FPGrabFocusTransient*, *FPOverrideGrabFocus* / *!FPOverrideGrabFocus*, *FPReleaseFocus* / *!FPReleaseFocus*, *!FPReleaseFocusTransient* / *FPReleaseFocusTransient*, *FPOverrideReleaseFocus* / *!FPOverrideReleaseFocus*, *StartsLowered* / *StartsRaised*, *IgnoreRestack* / *AllowRestack*, *FixedPosition* / *VariablePosition*, *FixedUSPosition* / *VariableUSPosition*, *FixedPPosition* / *VariablePPosition*, *FixedSize* / *VariableSize*, *FixedUSSize* / *VariableUSSize*, *FixedPSize* / *VariablePSize*, *!Closable* / *Closable*, *!Iconifiable* / *Iconifiable*, *!Maximizable* / *Maximizable*, *!AllowMaximizeFixedSize* / *AllowMaximizeFixedSize*, *IconOverride* / *NoIconOverride* / *NoActiveIconOverride*, *DepressableBorder* / *FirmBorder*, *MinWindowSize*, *MaxWindowSize*, *IconifyWindowGroups* / *IconifyWindowGroupsOff*, *ResizeOpaque* / *ResizeOutline*, *BackingStore* / *BackingStoreOff* / *BackingStoreWindowDefault*, *Opacity* / *ParentalRelativity*, *SaveUnder* / *SaveUnderOff*, *WindowShadeShrinks* / *WindowShadeScrolls*, *WindowShadeSteps*, *WindowShadeAlwaysLazy* / *WindowShadeBusy* / *WindowShadeLazy*, *EWMHDonateIcon* / *EWMHDontDonateIcon*, *EWMHDonateMiniIcon* / *EWMHDontDonateMiniIcon*, *EWMHMiniIconOverride* / *EWMHNoMiniIconOverride*, *EWMHUseStackingOrderHints* / *EWMHIgnoreStackingOrderHints*, *EWMHIgnoreStateHints* / *EWMHUseStateHints*, *EWMHIgnoreStrutHints* / *EWMHUseStrutHints*, *EWMHIgnoreWindowType* / *!EWMHIgnoreWindowType*, *EWMHMaximizeIgnoreWorkingArea* / *EWMHMaximizeUseWorkingArea* / *EWMHMaximizeUseDynamicWorkingArea*, *EWMHPlacementIgnoreWorkingArea* / *EWMHPlacementUseWorkingArea* / *EWMHPlacementUseDynamicWorkingArea*, *MoveByProgramMethod*, *Unmanaged*, *State*, *SnapGrid*, *SnapAttraction*, *EdgeMoveDelay*, *EdgeResizeDelay*. *EdgeMoveResistance*, *InitialMapCommand*

In the above list some options are listed as style−option/opposite−style−option. The opposite−style−option for entries that have them describes the fvwm default behavior and can be used if you want to change the fvwm default behavior.

Focus policy

ClickToFocus instructs fvwm to give the focus to a window when it is clicked in. The default *MouseFocus* (or its alias *FocusFollowsMouse*) tells fvwm to give a window the focus as soon as the pointer enters the window, and take it away when the pointer leaves the window. *SloppyFocus* is similar, but doesn't give up the focus if the pointer leaves the window to pass over the root window or a ClickToFocus window (unless you click on it, that is), which makes it possible to move the mouse out of the way without losing focus. A window with the style *NeverFocus* never receives the focus. This is useful for modules like **FvwmButtons**. for example. Note: Once any of the "FP..." styles has been used, the defaults that come with the basic focus policies are not restored when the latter are used again. For example, once !FPGrabFocus has been used, using ClickToFocus does not restore FPGrabFocus.

The focus model can be augmented with several additional options. In fvwm−2.5.3 and later, there are a large number of advanced options beginning with "FP" or "!FP". These options shall replace the older options one day and are described first. Using any of these new options may limit compatibility with older releases. In general, options beginning with "FP" turn a feature on, while those beginning with "!FP" turn it off.

Focusing the window

With *FPEnterToFocus*, when the pointer enters a window it receives focus.

With *FPLeaveToUnfocus* a window loses focus when the pointer leaves it.

With *FPClickToFocus*, *FPClickDecorToFocus* or *FPClickIconToFocus*, a window receives focus when the inside of the window or the decorations or its icon is clicked.

The *FPFocusByProgram* style allows windows to take the focus themselves.

The !*FPFocusByFunction* style forbids that a window receives the focus via the **Focus** and **FlipFocus** commands.

The *FPFocusByFunctionWarpPointer* style controls if the pointer is warped to a selected window when the **Focus** command is used.

FPLenient allows focus on windows that do not want it, like **FvwmPager** or xclock.

The *FPFocusClickButtons* style takes a list of mouse buttons that can be clicked to focus or raise a window when the appropriate style is used. The default is to use the first three buttons ("123").

The *FPFocusClickModifiers* style takes a list of modifier keys just like the **Key** command. The exact combination of modifier keys must be pressed for the click to focus or raise a window to work. The default is to use no modifiers ("N").

With the *FPPassFocusClick* style, the click that was used to focus a window is passed to the application.

With the *FPAllowFocusClickFunction* style, the click that was used to focus a window can also trigger a normal action that was bound to the window with the **Mouse** command).

If the *FPIgnoreFocusClickMotion* style is used, clicking in a window and then dragging the pointer with the button held down does not count as the click to focus the window. Instead, the application processes these events normally. This is useful to select text in a terminal window with the mouse without raising the window. However, mouse bindings on the client window are not guaranteed to work anymore (see **Mouse** command). This style forces the initial click to be passed to the application. The distance that the pointer must be moved to trigger this is controlled by the **MoveThreshold** command.

The *FPSortWindowlistByFocus* and !*FPSortWindowlistByFocus* styles control whether the internal window list is sorted in the order the windows were focused or in the order they were created. The latter is the default for *ClickToFocus* and *SloppyFocus*.

Clicking the window to raise

The styles *FPClickRaisesFocused*, *FPClickDecorRaisesFocused* and *FPClickIconRaisesFocused* allow one to raise the window when the interior or the decorations or the icon of the window is clicked while the window is already focused.

The styles *FPClickRaisesUnfocused*, *FPClickDecorRaisesUnfocused* and *FPClickIconRaisesUnfocused* allow one to raise the window when the interior or the decorations or the icon of the window is clicked while the window is not yet focused.

With the *FPPassRaiseClick* style, the click that was used to raise the window is passed to the application.

With the *FPAllowRaiseClickFunction* style, the click that was used to raise the window can also trigger a normal action that was bound to the window with the **Mouse** command.

If the *FPIgnoreRaiseClickMotion* style is used, clicking in a window and then dragging the pointer with the button held down does not count as the click to raise the window. Instead, the application processes these events normally. This is useful to select text in a terminal window with the mouse without raising the window. However, mouse bindings on the client window are not guaranteed to work anymore (see **Mouse** command. Note that this style forces that the initial click is passed to the application. The distance that the pointer must be moved to trigger this is controlled by the **MoveThreshold** command.

Grabbing the focus when a new window is created

New normal or transient windows with the *FPGrabFocus* or *FPGrabFocusTransient* style automatically receive the focus when they are created. *FPGrabFocus* is the default for windows with the *ClickToFocus* style. Note that even if these styles are disabled, the application may take the focus itself. Fvwm can not prevent this.

The *OverrideGrabFocus* style instructs fvwm to never take away the focus from such a

window via the *GrabFocus* or *GrabFocusTransient* styles. This can be useful if you like to have transient windows receive the focus immediately, for example in a web browser, but not while you are working in a terminal window or a text processor.

The above three styles are accompanied by *FPReleaseFocus*, *FPReleaseFocusTransient* and *FPOverrideReleaseFocus*. These control if the focus is returned to another window when the window is closed. Otherwise no window or the window under the pointer receives the focus.

ClickToFocusPassesClickOff and *ClickToFocusPassesClick* controls whether a mouse click to focus a window is sent to the application or not. Similarly, *ClickToFocusRaisesOff*/*MouseFocusClickRaisesOff* and *ClickToFocusRaises*/*MouseFocusClickRaises* control if the window is raised (but depending on the focus model).

Note: in fvwm versions prior to 2.5.3, the "Click..." options applied only to windows with *ClickToFocus* while the "Mouse..." options applied to windows with a different focus policy. This is no longer the case.

The old *GrabFocus* style is equivalent to using *FPGrabFocus* + *FPReleaseFocus*.

The old *GrabFocusTransient* style is equivalent to using *FPGrabFocusTransient* + *FPReleaseFocusTransient*.

Lenience is equivalent to the new style *FPLenient*.

Window title

The *Title* and !Title options determine if the window has a title−bar or not. By default all windows have a title−bar. *NoTitle* is equivalent to *!Title* but is deprecated.

Windows with the *TitleAtBottom*, *TitleAtLeft* or *TitleAtRight* style have a title−bar below, to the left or to the right of the window instead of above as usual. The *TitleAtTop* style restores the default placement. Even if the window has the *!Title* style set, this affects the **WindowShade** command. Please check the **WindowShade** command for interactions between that command and these styles. Titles on the left or right side of the windows are augmented by the following styles:

Normally, the text in titles on the left side of a window is rotated counterclockwise by 90 degrees from the normal upright position and 90 degrees clockwise for titles on the right side. It can also be rotated in the opposite directions with*LeftT itleRotatedCW* if *TitleAtLeft* is used, and with *RightTitleRotatedCCW* if *TitleAtRight* is used. The defaults can be restored with *LeftTitleRotatedCCW* and *RightTitleRotatedCW*. A normal horizontal text may be rotated as well with *TopTitleRotated* if *TitleAtTop* is used, and with *BottomTitleRotated* if *TitleAtBottom* is used. The defaults can be restored with *TopTitleNotRotated* and *BottomTitleNotRotated*.

By default the title bar decoration defined using the **TitleStyle** command is rotated following the title text rotation (see the previous paragraph). This can be disabled by using the !*UseTitleDecorRotation* style. *UseTitleDecorRotation* reverts back to the default.

With the *StippledTitle* style, titles are drawn with the same effect that is usually reserved for windows with the *Sticky*, *StickyAcrossPages* or *StickyAcrossDesks* style. *!StippledTitle* reverts back to normal titles. *StippledTitleOff* is equivalent to *!StippledTitle* but is deprecated.

Color takes two arguments. The first is the window−label text color and the second is the window decorations normal background color. The two colors are separated with a slash. If the use of a slash causes problems then the separate *ForeColor* and *BackColor* options can be used.

Colorset takes the colorset number as its sole argument and overrides the colors set by

Color. Instead, the corresponding colors from the given colorset are used. Note that all other features of a colorset are not used. Use the **Colorset** decoration style in the **TitleStyle** and *ButtonStyle* command for that. To stop using the colorset, the colorset number is omitted.

The *HilightFore*, *HilightBack* and *HilightColorset* style options work exactly like *ForeColor*, *BackColor* and *Colorset* but are used only if the window has the focus. These styles replace the old commands **HilightColor** and *HilightColorset*.

BorderColorset takes the colorset number as its sole argument and overrides the colors set by *Color* or *Colorset*. for the window border. To stop using a colorset, the argument is omitted.

The *HilightBorderColorset* style option works similarly to *BorderColorset* but is used when the window has the focus.

!*IconTitle* disables displaying icon labels while the opposite style *IconTitle* enables icon labels (default behaviour). *NoIconTitle* is equivalent to *!IconTitle* but is deprecated.

IconTitleColorset takes the colorset number as its sole argument and overrides the colors set by *Color* or *Colorset*. To stop using this colorset, the argument is omitted.

HilightIconTitleColorset takes the colorset number as its sole argument and overrides the colors set by **HilightColor** or *HilightColorset*. To stop using this colorset, the argument is omitted.

IconBackgroundColorset takes the colorset number as its sole argument and uses it to set a background for the icon picture. By default the icon picture is not drawn onto a background image. To restore the default, the argument is omitted.

IconTitleRelief takes one numeric argument that may be between −50 and +50 pixels and defines the thickness of the 3D relief drawn around the icon title. With negative values the icon title gets a pressed in look. The default is 2 and it is restored if the argument is omitted.

IconBackgroundRelief takes one numeric argument that may be between −50 and +50 pixels and defines the thickness of the 3D relief drawn around the icon picture background (if any). With negative values the icon background gets a pressed in look. The default is 2 and it is restored if the argument is omitted.

IconBackgroundPadding takes one numeric argument that may be between 0 and 50 pixels and defines the amount of free space between the relief of the icon background picture (if any) and the icon picture. The default is 2 and it is restored if the argument is omitted.

The *Font* and *IconFont* options take the name of a font as their sole argument. This font is used in the window or icon title. By default the font given in the **DefaultFont** command is used. To revert back to the default, use the style without the name argument. These styles replace the older **WindowFont** and *IconFont* commands.

The deprecated *IndexedWindowName* style causes fvwm to use window titles in the form

name (i)

where *name* is the exact window name and *i* is an integer which represents the *i th* window with *name* as window name. This has been replaced with:

TitleFormat %n (%t)

ExactWindowName restores the default which is to use the exact window name. Deprecated in favour of:

TitleFormat %n

IndexedIconName and *ExactIconName* work the same as *IndexedWindowName* and *ExactWindowName* styles but for the icon titles. Both are deprecated in favour of:

```
IconTitleFormat %n (%t)
IconTitleFormat %n
```

TitleFormat describes what the visible name of a window should look like, with the following placeholders being valid:

%n
Insert the window's name.

%i
Insert the window's icon name.

%c
Insert the window's class name.

%r
Insert the window's resource name.

%t
Insert the window count.

%I
Insert the window ID.

%%
Insert a literal '%' character.

Any amount of whitespace may be used, along with other characters to make up the string −− but a valid *TitleFormat* string must contain at least one of the placeholders mentioned. No quote stripping is performed on the string, so for example the following is printed verbatim:

```
TitleFormat " %n " −> [%t] −>      [%c]
```

Note: It's perfectly possible to use a *TitleFormat* which can result in wiping out the visible title altogether. For example:

```
TitleFormat %z
```

Simply because the placeholder '%z' isn't supported. This is not a bug but rather a facet of how the formatting parser works.

IconTitleFormat describes what the visible icon name of a window should look like, with the options being the same as *TitleFormat*.

Title buttons

Button and !*Button* take a numeric argument which is the number of the title−bar button which is to be shown or omitted. *NoButton* is equivalent to *!Button* but is deprecated.

MwmButtons makes the **Maximize** button look pressed−in when the window is maximized. See the*MwmDecorMax* flag in **ButtonStyle** for more information. T o switch this style off again, use the **FvwmButtons** style.

Borders

!*Borders* suppresses the window border (but not the title) completely. The*Bor ders* style enables them again. Without borders, all other styles affecting window borders are meaningless.

MwmBorder makes the 3D bevel more closely match Mwm's. *FvwmBorder* turns off the previous option.

With the !*Handles* style, the window does not get the handles in the window corners that

are commonly used to resize it. With *!Handles*, the width from the *BorderWidth* style is used. By default, or if *Handles* is specified, the width from the *HandleWidth* style is used. *NoHandles* is equivalent to *!Handles* but is deprecated.

HandleWidth takes a numeric argument which is the width of the border to place the window if it does have resize−handles. Using HandleWidth without an argument restores the default.

BorderWidth takes a numeric argument which is the width of the border to place the window if it does not have resize−handles. It is used only if the *!Handles* style is specified too. Using BorderWidth without an argument restores the default.

DepressableBorder makes the border parts of the window decoration look sunken in when a button is pressed over them. This can be disabled again with the *FirmBorder* style.

Icons, shading, maximizing, movement, resizing

Icon takes an (optional) unquoted string argument which is the icon bitmap or pixmap to use. Icons specified this way override pixmap icons, but not icon windows or the ewmh icon, provided by the client in the application (with the WM_HINTS property or with the ewmh _NET_WM_ICON property). The *IconOverride* style changes the behavior to override any client−provided icons; the *NoIconOverride* style changes the behavior to not override any client−provided icons; the default overriding behavior can be activated with the *NoActiveIconOverride* style. With this style, fvwm uses application provided icons if the icon is changed but uses the icon provided in the configuration file until then.

There is one exception to these rules, namely

```
Style * Icon unknown.xpm
```

doesn't force the unknown.xpm icon on every window, it just sets the default icon like the DefaultIcon command. If you really want all windows to have the same icon, you can use

```
Style ** Icon unknown.xpm
```

If the *NoIcon* attribute is set then the specified window simply disappears when it is iconified. The window can be recovered through the window−list. If*Icon* is set without an argument then the *NoIcon* attribute is cleared but no icon is specified. An example which allows only the **FvwmPager** module icon to exist:

```
Style * NoIcon
Style FvwmPager Icon
```

IconBox takes no argument, four numeric arguments (plus optionally a screen specification), an X11 geometry string or the string "none":

```
IconBox [screen scr−spec] l t r b
```

or

```
IconBox geometry
```

Where *l* is the left coordinate, *t* is the top, *r* is right and *b* is bottom. Negative coordinates indicate distance from the right or bottom of the screen. If the first argument is the word *screen*, the *scr-spec* argument specifies the Xinerama screen on which the IconBox is defined. It can be the usual screen Xinerama specification, 'p', ´c', 'g', a screen number or the additional 'w' for the screen where the window center is located. This is only useful with multiple Xinerama screens. The "l t r b" specification is more flexible than an X11 geometry. For example:

IconBox −80 240 −1 −1

defines a box that is 80 pixels wide from the right edge, 240 pixels down from the top, and continues to the bottom of the screen.

Perhaps it is easier to use is an X11 geometry string though:

IconBox 1000x70−1−1

places an 1000 by 70 pixel icon box on the bottom of the screen starting in the lower right hand corner of the screen. One way to figure out a geometry like this is to use a window that resizes in pixel increments, for example, xv. Then resize and place the xv window where you want the iconbox. Then use FvwmIdent to read the windows geometry. The icon box is a region of the screen where fvwm attempts to put icons for any matching window, as long as they do not overlap other icons. Multiple icon boxes can be defined as overflow areas. When the first icon box is full, the second one is filled. All the icon boxes for one style must be defined in one **Style** command. For example:

Style * IconBox −80 240 −1 −1, \
IconBox 1000x70−1−1

A Style command with the IconBox option replaces any icon box defined previously by another Style command for the same style. That's why the backslash in the previous example is required.

Note: The geometry for the icon box command takes the additional screen specifier "@w" in case a Xinerama setup is used. This designates the screen where the window center is located. The additional screen specifier is not allowed anywhere else.

If you never define an icon box, or you fill all the icon boxes, fvwm has a default icon box that covers the screen, it fills top to bottom, then left to right, and has an 80x80 pixel grid. To disable all but the default icon box you can use IconBox without arguments in a separate **Style** command. To disable all icon boxes including the default icon box, the argument "none" can be specified.

Hint: You can auto arrange your icons in the icon box with a simple fvwm function. Put the "DeiconifyAndRearrange" function below in your configuration file:

AddToFunc DeiconifyAndRearrange
+ C **Iconify** off
+ C **All** (CurrentPage, Iconic) **PlaceAgain** Icon

And then replace all places where you call the **Iconify** command to de−iconify an icon with a call to the new function. For example replace

AddToFunc IconFunc
+ C **Iconify** off
+ M **Raise**
+ M **Move**
+ D **Iconify** off

Mouse 1 I A **Iconify** off

with

AddToFunc IconFunc
+ C DeiconifyAndRearrange
+ M **Raise**
+ M **Move**
+ D DeiconifyAndRearrange

Mouse 1 I A DeiconifyAndRearrange

IconGrid takes 2 numeric arguments greater than zero.

IconGrid x y

Icons are placed in an icon box by stepping through the icon box using the *x* and *y* values for the icon grid, looking for a free space. The default grid is 3 by 3 pixels which gives a tightly packed appearance. To get a more regular appearance use a grid larger than your largest icon. Use the *IconSize* argument to clip or stretch an icon to a maximum size. An *IconGrid* definition must follow the **IconBox** definition that it applies to:

Style * IconBox −80x240−1−1, IconGrid 90 90

IconFill takes 2 arguments.

IconFill *Bottom Right*

Icons are placed in an icon box by stepping through the icon box using these arguments to control the direction the box is filled in. By default the direction is left to right, then top to bottom. This would be expressed as:

IconFill left top

To fill an icon box in columns instead of rows, specify the vertical direction (top or bottom) first. The directions can be abbreviated or spelled out as follows: "t", "top", "b", "bot", "bottom", "l", "lft", "left", "r", "rgt", "right". An **IconFill** definition must follow the **IconBox** definition that it applies to:

Style * IconBox −80x240−1−1, IconFill b r

IconSize sets limits on the size of an icon image. Both user−provided and application−provided icon images are affected.

IconSize *[* width height *[* maxwidth maxheight *]]*

All arguments are measured in pixels. When all four arguments are passed to *IconSize*, *width* and *height* represent the minimum size of an icon, and *maxwidth* and *maxheight* represent the maximum size of an icon. Icon images that are smaller than the minimum size are padded. Icon images that are bigger than the maximum size are clipped.

If only two arguments are passed to *IconSize*, *width* and *height* represent the absolute size of an icon. Icons covered by this style are padded or clipped to achieve the given size.

If no arguments are specified, the default values are used for each dimension. This effectively places no limits on the size of an icon.

The value of "−1" can be used in place of any of the arguments to specify the default value for that dimension.

In addition to the numeric arguments, 1 additional argument can be "Stretched", "Adjusted", or "Shrunk".

Note that module provided icon managers are not affected by this style.

MiniIcon specifies a pixmap to use as the miniature icon for the window. This miniature icon can be drawn in a title−bar button (see **ButtonStyle**), and can be used by various fvwm modules (**FvwmIconMan** and **FvwmPager**). It takes the name of a pixmap as an argument.

WindowShadeShrinks and *WindowShadeScrolls* control if the contents of a window that is being shaded with the **WindowShade** command are scrolled (default) or if they stay in place. The shrinking mode is a bit faster

The *WindowShadeSteps* option selects the number of steps for animation when shading a window with **WindowShade**. It takes one number as its argument. If the number has a trailing '*p*' it sets the number of pixels to use as the step size instead of a fixed number of steps. 0 disables the animation. This happens too if the argument is omitted or invalid.

The **WindowShade** command has two modes of operation: busy and lazy shading. Busy shading can be 50% slower than lazy shading, but the latter can look strange under some conditions, for example, if the window borders, buttons or the title are filled with a tiled pixmap. Also, the window handles are not drawn in lazy mode and the border relief may only be drawn partially right before the window reaches the shaded state or tight after leaves the unshaded state. By default, fvwm uses lazy mode if there are no bad visual effects (not counting the window handles) and busy mode otherwise. Use the *WindowShadeAlwaysLazy or WindowShadeBusy* to force using the lazy or busy mode. The default setting is restored with *WindowShadeLazy*.

ResizeOpaque instructs fvwm to resize the corresponding windows with their contents visible instead of using an outline. Since this causes the application to redraw frequently it can be quite slow and make the window flicker excessively, depending on the amount of graphics the application redraws. The*ResizeOutline* style (def ault) negates the *ResizeOpaque* style. Many applications do not like their windows being resized opaque, e.g. XEmacs, Netscape or terminals with a pixmap background. If you do not like the result, do not use the *ResizeOpaque* style for these windows. To exempt certain windows from opaque resizing you could use these lines in your configuration file:

```
Style * ResizeOpaque
Style rxvt ResizeOutline
Style emacs ResizeOutline
```

Sticky makes the window sticky, i.e. it is always visible on each page and each desk. The opposite style, *Slippery* reverts back to the default.

StickyIcon makes the window sticky when it's iconified. It de−iconifies on top the active desktop. *SlipperyIcon* reverts back to the default.

StickyAcrossPages and *StickyAcrossPagesIcon* work like *Sticky* and *StickyIcon*, but stick the window only across pages, not desks while *StickyAcrossDesks and StickyAcrossDesksIcon* works the other way round.

Windows that have been marked as *Sticky* or *StickyAcrossDesks* or *StickyAcrossPages* will have stipples drawn on the titlebar. This can be negated with the !*StickyStippledTitle* style. The style *StickyStippledTitle* puts back the stipples where that window has also been marked as *Sticky*. Note that this is the default style for *Sticky* windows. Sticky icons will have stipples drawn on the icon title. This can be disabled in the same way with the !*StickyStippledIconTitle* style.

Windows with the *StartIconic* style are shown as icons initially. Note that some applications counteract that by deiconifying themselves. The default is to not iconify windows and can be set with the *StartNormal* style.

StickyIcon makes the window sticky when it's iconified. It de−iconifies on top the active desktop. *SlipperyIcon* reverts back to the default.

StickyIconPage works like *StickyIcon*, but sticks the icon only across pages, not desks while *StickyIconDesk* works the other way round.

StippledIconTitle works like *StippledTitle* in that it draws stipples on the titles of icons but doesn't make the icon sticky.

IgnoreRestack makes fvwm ignore attempts of clients to raise or lower their own windows. By default, the opposite style, *AllowRestack* is active.

FixedPosition and *FixedUSPosition* make fvwm ignore attempts of the user to move the window. It is still possible to move the window by resizing it. To allow the user to move windows, use the *VariablePosition* or *VariableUSPosition* style.

FixedSize and *FixedUSSize* make fvwm ignore attempts of the user to resize the window. To allow the user to resize windows, use the *VariableSize* or *VariableUSSize* style.

FixedPPosition and *FixedPSize* make fvwm ignore attempts of the program to move or resize its windows. To allow this kind of actions, use the *VariablePPosition* or *VariablePSize* style. These styles may sometimes affect the initial placement and dimensions of new windows (depending on the application). If windows are created at strange places, try either the *VariablePPosition* or *!UsePPosition* styles. The *FixedPSize* style may screw up window dimensions for some applications. Do Not use this style in this case.

MoveByProgramMethod affects how fvwm reacts to requests by the application to move its windows. By default, fvwm tries to detect which method to use, but it sometimes detects the wrong method. You may come across a window that travels across the screen by a few pixels when the application resizes it, moves to a screen border with the frame decorations off screen, that remembers its position for the next time it starts but appears in a slighly shifted position, or that attepmts to become full screen but has the. Try out both options, *UseGravity* and *IgnoreGravity* on the window (and that window only) and see if that helps. By default, fvwm uses the *AutoDetect* method. Once the method was detected, it is never changed again. As long as fvwm can not detect the proper method, it uses *IgnoreGravity*. To force fvwm to retry the detection, use one of the other two options first and then use *AutoDetect* again.

Note: This option was introduced to alleviate a problem with the ICCCM specification. The ICCCM clearly states that the *UseGravity* option should be used, but traditionally applications ignored this rule.

Closable enables the functions **Close**, **Delete** and **Destroy** to be performed on the windows. This is on by default. The opposite,*!Closable*, inhibits the windo w to be closed.

Iconifiable enables the function **Iconify** to be performed on the windows. This is on by default. The opposite, *!Iconifiable*, inhibits the window from being iconified.

Maximizable enables the function **Maximize** to be performed on the windows. This is on by default. The opposite,*!Maximizable*, inhibits the windo w from being maximized.

AllowMaximizeFixedSize enables the function **Maximize** to be performed on windows that are not resizable, unless maximization has been disabled either using the style *!Maximizable* or through WM hints. This is on by default. The opposite,*!AllowMaximizeF ixedSize*, inhibits all windows that are not resizable from being maximized.

ResizeHintOverride instructs fvwm to ignore the program supplied minimum and maximum size as well as the resize step size (the character size in many applications). This can be handy for broken applications that refuse to be resized. Do not use it if you do not need it. The default (opposite) style is *NoResizeOverride*.

MinWindowSize [width [p] height [p]] Tells fvwm the minimum width and height of a window. The values are the percentage of the total screen area. If the letter '*p*' is appended to either of the values, the numbers are interpreted as pixels. This command is useful for certain versions of xemacs which freak out if their windows become too small. If you omit he parameters or their values are invalid, both limits are set to 0 pixels (which is the default value).

MaxWindowSize [width [p] height [p]] Tells fvwm the maximum width and height of a window. The values are the percentage of the total screen area. If the letter '*p*' is appended to either of the values, the numbers are interpreted as pixels. This command is useful to force large application windows to be fully visible. Neither *height* nor *width* may be less than 100 pixels. If you omit the parameters or their values are invalid, both limits are set to 32767 pixels (which is the default).

With *IconifyWindowGroups* all windows in the same window group are iconified and deiconified at once when any window in the group is (de)iconified. The default is *IconifyWindowGroupsOff*, which disables this behavior. Although a number of applications use the window group hint, it is rarely used in a proper way, so it is probably best to use *IconifyWindowGroups* only for selected

applications.

The option *SnapAttraction* affects interactive window movement: If during an interactive move the window or icon comes within *proximity* pixels of another the window or icon, it is moved to make the borders adjoin. The default of 0 means that no snapping happens. Calling this command without arguments turns off snap attraction and restores the default behavior. Please refer also to the **SnapGrid** command.

The second argument determined is optional and may be set to one of the five following values: With *All* both icons and windows snap to other windows and other icons. *SameType* lets windows snap only to windows, and icons snap only to icons. With *Windows* windows snap only to other windows. Similarly with*Icons* icons snap only to other icons. W ith *None* no snapping takes place. This option can be useful in conjunction with the following argument if you only want to snap against the screen edges. The default behavior is *All*.

The third and last optional argument may be set to one of the four following values:

- With *Screen* the already snapping icons or windows, which is controlled by the second argument, will snap now also to the screen edges.
- *ScreenWindows* snaps only windows to the screen edges.
- *ScreenIcons* snaps only icons to the screen edges.
- *ScreenAll* snaps windows and icons to the screen edges.

The option *SnapGrid* defines an invisible grid on the screen. During an interactive move a window or icon is positioned such that its location (top left corner) is coincident with the nearest grid point. The default *x−grid−size* and *y−grid−size* setting are both 1, which is effectively no grid all.

An interactive move with both **SnapGrid** and *SnapAttraction* results in the window being moved to be adjacent to the nearest window border (if within snap proximity) or grid position. The window moves the shortest distance possible to satisfy both **SnapGrid** and *SnapAttraction*. Note that the x and y coordinates are not coupled. For example, a window may snap to another window on the x axis while snapping to a grid point on the y axis. Using this style without arguments reinstates the default settings.

The styles *EdgeMoveDelay* and *EdgeResizeDelay* tells how hard it should be to change the desktop viewport by moving or resizing a window over the edge of the screen. The parameter tells how many milliseconds the pointer must spend on the screen edge before fvwm moves the viewport. The command**EdgeScr oll** determines how far the viewport is scrolled. If −1 is given as the delay, page flipping is disabled completely. The defaults are no delay for moving (0) and no flipping for resizing (−1). Using these styles without any argument restores the default settings. Note that, with

EdgeScroll 0 0

it is still possible to move or resize windows across the edge of the current screen. See also **EdgeThickness**.

The option *EdgeMoveResistance* makes it easier to place a window directly adjacent to the screen's or xinerama screen's border. It takes one or two parameters. The first parameter tells how many pixels over the edge of the screen a window's edge must move before it actually moves partially off the screen. The optional second parameter does the same as the first, but for individual Xinerama screens. If omitted, the value of the first parameter is assumed for this type of movement. Set the second parameter to 0 to zero to ignore individual xinerama screen edges. Note that the center of the window being moved determines the xinerama screen on which the window should be kept. Both values are 0 by default. To restore the defaults, the option

EdgeMoveResistance can be used without any parameters.

The option *InitialMapCommand* allows for any valid fvwm command or function to run when the window is initially mapped by fvwm. Example:

Style MyWindow StartsOnPage 0 0, InitialMapCommand Iconify

This would hence place the window called *MyWindow* on page 0 0 for the current desk, and immediately run the **Iconify** command on that window.

Note that should *InitialMapCommand* be used as a global option for all windows, but there is a need that some windows should not have this command applied, then an action of **Nop** can be used on those windows, as in the following example:

Style * InitialMapCommand Iconify
Style XTeddy InitialMapCommand Nop

Window Manager placement

Applications can place windows at a particular spot on the screen either by window manager hints or a geometry specification. When they do neither, then the window manager steps in to find a place for the window. Fvwm knows several ways to deal with this situation. The default is *TileCascadePlacement*.

PositionPlacement [*Center*|*UnderMouse*|*move−arguments*] When used without an argument, new windows are placed in the top left corner of the display. With the argument *Center*, all new window appear at the center of the screen, and with *UnderMouse*, windows are centered under the mouse pointer where possible. If the window is unable to fit on the screen because the pointer is at the edge of the screen, then the window is forced on−screen using this option. If any other *move−arguments* are given, they are interpreted exactly as the **Move** command does (with the exception that references to the current window position do not work as the window has not been placed yet).

CascadePlacement automatically place new windows in a cascading fashion.

TileCascadePlacement automatically places new windows in a smart location − a location in which they do not overlap any other windows on the screen. If no such position can be found *CascadePlacement* is used as a fall−back method.

TileManualPlacement This is the same as *TileCascadePlacement*, but uses *ManualPlacement* as the fall−back method.

MinOverlapPlacement automatically places new windows in a location in which the overlapping area in pixels of other windows is minimized. By default this placement policy tries to avoid overlapping icons and windows on higher layers. This can be configured with the *MinOverlapPlacementPenalties* style.

MinOverlapPercentPlacement is similar to *MinOverlapPlacement* but tries to minimize the overlapped percentages of other windows instead of the overlapped area in pixels. This placement policy tries to avoid covering other windows completely and tries even harder not to cover small windows. This can be configured with the*MinOverlapPlacementP enalties* and *MinOverlapPercentPlacementPenalties* styles.

MinOverlapPlacementPenalties takes at most 6 positive or null decimal arguments:

normal ontop icon sticky below strut

if trailing arguments are missing the default is used which is:

1 5 10 1 0.05 50

To reset this style to the default values, prefix it with a '!'. This style configures the *MinOverlapPlacement* and *MinOverlapPercentPlacement* placement policy. The*normal* f actor

affects normal windows, the *ontop* factor affects windows with a greater layer than the window being placed, the *icon* factor affects icons, the *sticky* factor affects sticky windows, the *below* factor affects windows with a smaller layer than the window being placed, the *strut* factor affects the complement of the EWMH working area if the window being placed has the *EWMHPlacementUseWorkingArea* style and windows with an EWMH strut hint (i.e., a "please do not cover me" hint) if the window being placed has the *EWMHPlacementUseDynamicWorkingArea* style. These factors represent the amount of area that these types of windows (or area) are counted as, when a new window is placed. For example, by default the area of ontop windows is counted 5 times as much as normal windows. So *MinOverlapPlacement* and *MinOverlapPercentPlacement* covers 5 times as much area of another window before it will cover an ontop window. To treat ontop windows the same as other windows, set this to 1. To really, really avoid putting windows under ontop windows, set this to a high value, say 1000. This style affects the window already mapped and not the window which is currently placed. There is one exception to this rule: in the case of the window being placed has the *EWMHPlacementUseWorkingArea* style the *strut* factor affects the placed window.

MinOverlapPercentPlacementPenalties takes at most 4 positive or null integer arguments:

cover_100 cover_95 cover_85 cover_75

if trailing arguments are missing the defaults are used which are:

12 6 4 1

To reset this style to the default values, prefix it with a '!'. This style affects the *MinOverlapPercentPlacement* placement policy and is similar to the *MinOverlapPlacementPenalties* style. The *cover_xx* factor is used when the window being placed covers at least *xx* percent of the window. This factor is added to the factor determined by the *MinOverlapPlacementPenalties* style.

ManualPlacement (aka active placement). The user is required to place every new window manually. The window only shows as a rubber band until a place is selected manually. The window is placed when a mouse button or any key except *Escape* is pressed. Escape aborts manual placement which places the window in the top left corner of the screen. If mouse button 2 is pressed during the initial placement of a window (respectively *Shift* and mouse button 1 in case Mwm emulation has been enabled with the **Emulate** command), the user is asked to resize the window too.

It is possible to define buttons usable to place windows with the **Move** command and the special context 'P' for placement (see **Move** command). However, you can't redefine the way to also resize the window other than the way it is affected by the **Emulate** command. The button used for placing the window can be checked with the *PlacedByButton* condition (see **Current** command).

Example:

```
Style * ManualPlacement

*FvwmEvent: PassID
*FvwmEvent: add_window GrowDownFunc
AddToFunc StartFunction
+ I FvwmEvent

AddToFunc GrowDownFunc
+ I windowid $0 (PlacedByButton 3) \
  Resize bottomright keep -0p
```

Now, whenever a window is created and the user presses button 3 to finish initial placement, the window is automatically enlarged until it hits the bottom screen border.

Old placement styles DumbPlacement / SmartPlacement / SmartPlacementOff, CleverPlacement / CleverPlacementOff, ActivePlacement / RandomPlacement, ActivePlacementsHonorsStartsOnPage / ActivePlacementsHonorsStartsOnPageOff, GlobalOpts SmartPlacementIsReallySmart / GlobalOpts SmartPlacementIsNormal are still supported but will be removed in the future. The old and new styles can be translated according to the following table:

GlobalOpts SmartPlacementIsReallySmart
Style * SmartPlacement
−−>
Style * SmartPlacement, CleverPlacement

GlobalOpts SmartPlacementIsNormal
Style * SmartPlacement
−−>
Style * SmartPlacement, CleverPlacementOff

Style * DumbPlacement, RandomPlacement
−−>
Style * CascadePlacement

Style * DumbPlacement, ActivePlacement
−−>
Style * ManualPlacement

Style * SmartPlacement, \
RandomPlacement, CleverPlacementOff
−−>
Style * TileCascadePlacement

Style * SmartPlacement, \
ActivePlacement, CleverPlacementOff
−−>
Style * TileManualPlacement

Style * SmartPlacement, CleverPlacement
−−>
Style * MinOverlapPlacement

Style * SmartPlacement, \
ActivePlacement, CleverPlacement
−−>
Style * MinOverlapPercentPlacement

Style * ActivePlacementsHonorsStartsOnPage
−−>
Style * ManualPlacementsHonorsStartsOnPage

Style * ActivePlacementsHonorsStartsOnPageOff
−−>
Style * ManualPlacementsHonorsStartsOnPageOff

Placement policy options and window stacking

!UsePPosition instructs fvwm to ignore the program specified position (PPosition hint) when adding new windows. Using PPosition is required for some applications, but if you do not have

one of those it's a real headache. Many programs set PPosition to something obnoxious like 0,0 (upper left corner). Note: *!UsePPosition* is equivalent to the deprecated option *!UsePPosition*

!UseUSPosition works like *!UsePPosition* but applies suppresses using the user specified position indicated by the program (USPosition hint). It is generally a bad thing to override the user's choice, but some applications misuse the USPosition hint to force their windows to a certain spot on the screen without the user's consent. Note: *!UseUSPosition* is equivalent to the deprecated option *!USPosition*

NoUseTransientPPosition and *UseTransientPPosition* work like *!UsePPosition* and *UsePPosition* but apply only to transient windows. Note:*!UseT ransientPPosition* is equivalent to the deprecated option *!TransientPPosition*

NoUseIconPosition instructs fvwm to ignore the program specified icon position (IconPosition hint) when iconifying the window. Note:*!UseIconP osition* is equivalent to the deprecated option *!IconPosition*

StartsOnDesk takes a numeric argument which is the desktop number on which the window should be initially placed. Note that standard Xt programs can also specify this via a resource (e.g. "−xrm '*Desk: 1'").

StartsOnPage takes 1, 2, or 3 numeric arguments. If one or three arguments are given, the first (or only) argument is the desktop number. If three arguments are given, the 2nd and 3rd arguments identify the x,y page position on the virtual window. If two arguments are given, they specify the page position, and indicate no desk preference. If only one argument is given, *StartsOnPage* functions exactly like *StartsOnDesk*. For those standard Xt programs which understand this usage, the starting desk/page can also be specified via a resource (e.g., "−xrm '*page: 1 0 2'"). *StartsOnPage* in conjunction with *SkipMapping* is a useful technique when you want to start an app on some other page and continue with what you were doing, rather than waiting for it to appear.

StartsOnScreen takes one argument. It can be 'p' for the primary screen, 'c' for the current screen (containing the mouse pointer), 'g' for the global screen or the screen number itself (counting from zero). A new window is placed on the specified Xinerama screen. The default is to place windows on the screen that contains the mouse pointer at the time the window is created. However, those windows which are not placed by fvwm (i.e., those with a USPosition hint from a user specified geometry) are normally placed in a position relative to the global screen. The *StartsOnScreen* style is also useful to cause these windows to be placed relative to a specific Xinerama screen. For example:

```
Style * StartsOnScreen c
```

Would cause all windows, including those with their own geometry to be placed relative to the current Xinerama screen rather than the global screen. For those standard Xt programs which understand this usage, the starting desk/page can also be specified via a resource (e.g., "−xrm '*fvwmscreen: c'"). ('fvwmscreen' was chosen because some applications already use ´.screen' for other purposes.)

StartsOnPageIncludesTransients causes the *StartsOnPage* style to be applied even for transient windows. This is not usually useful, since transients are usually pop ups that you want to appear in your visible viewport; but occasionally an application uses a transient for something like a startup window that needs to be coerced into place.

ManualPlacementIgnoresStartsOnPage suppresses *StartsOnPage* or *StartsOnDesk* placement in the event that both *ManualPlacement* and *SkipMapping* are in effect when a window is created. This prevents you from interactively placing a window and then wondering where it disappeared to, because it got placed on a different desk or page. *ManualPlacementHonorsStartsOnPage* allows this to happen anyway. The option has no effect if *SkipMapping* is not in effect, because fvwm switches to the proper desk/page to perform interactive placement. The default is *ManualPlacementIgnoresStartsOnPage*; *ManualPlacementHonorsStartsOnPage* matches the way

the old *StartsOnDesk* style used to handle the situation.

CaptureHonorsStartsOnPage causes the initial capture (of an already existing window) at startup to place the window according to the *StartsOnPage* and *StartsOnScreen* desk, page and Xinerama screen specification. *CaptureIgnoresStartsOnPage* causes fvwm to ignore these settings (including *StartsOnDesk*) on initial capture. The default is *CaptureIgnoresStartsOnPage*.

RecaptureHonorsStartsOnPage causes a window to be placed according to, or revert to, the *StartsOnPage* and *StartsOnScreen* desk, page and Xinerama screen specification on **Restart** or *Recapture*. *RecaptureIgnoresStartsOnPage* causes fvwm to respect the current window position on **Restart** or **Recapture**. The default is *RecaptureIgnoresStartsOnPage*.

Layer accepts one optional argument: a non−negative integer. This is the layer the window is put in. If no argument is given, any previously set value is deleted and the default layer is implied.

StaysOnTop puts the window in the top layer. This layer can be changed by the command **DefaultLayers**; the default is 6.

StaysPut puts the window in the put layer. This layer can be changed by the command **DefaultLayers**; the default is 4.

StaysOnBottom puts the window in the bottom layer. This layer can be changed by the command **DefaultLayers**; the default is 2.

StartsLowered instructs fvwm to put the window initially at the bottom of its layer rather than the default *StartsRaised*.

StartShaded tells fvwm to shade the window. An optional direction argument may be given, which can be one of "*North*", "*South*", "*West*", "*East*", "*NorthWest*", "*NorthEast*", "*SouthWest*", "*SouthEast*" or if no direction is given, the default is to shade north.

SkipMapping tells fvwm not to switch to the desk the window is on when it gets mapped initially (useful with *StartsOnDesk* or *StartsOnPage*).

KeepWindowGroupsOnDesk makes new windows that have the window group hint set appear on the same desk as the other windows of the same group. Since this behavior may be confusing, the default setting is *ScatterWindowGroups*. The window group hint is ignored when placing windows in this case.

Transient windows

DecorateTransient causes transient windows, which are normally left undecorated, to be given the usual fvwm decorations (title bar, buttons, etc.). Note that some pop−up windows, such as the xterm menus, are not managed by the window manager and still do not receive decorations. *NakedTransient* (the default) causes transient windows not to be given the standard decorations. You can only bind keys or mouse buttons to the sides and the client part of an undecorated window (’S’ and ´W’ contexts in bindings, see **Mouse** and *Key* commands).

A window with the *RaiseTransient* style that has transient windows raises all its transients when it is raised. The *DontRaiseTransient* style disables this behavior. All windows are then treated as if they had no transients.

A window with the *LowerTransient* style that has transient windows lowers all its transients when it is lowered. The*DontLowerT ransient* style disables this behavior. All windows are then treated as if they had no transients.

The *StackTransientParent* style augments *RaiseTransient* and *LowerTransient* styles. Raising a window with *StackTransientParent* style transfers the raise action to the main window if the window being raised is a transient and its main window has *RaiseTransient* style; this effect makes raise on a transient act just like raise on its main − the whole group is raised. Similar behavior holds for lowering a whole group of transients when the main has *LowerTransient* style. *DontStackTransientParent* turns this behavior off. *(Dont)StackTransientParent* has no effect if *RaiseTransient* and *LowerTransient* are not used.

A reasonable emulation of Motif raise/lower on transients is possible like this

```
Style * RaiseTransient
Style * LowerTransient
Style * StackTransientParent
```

Extended Window Manager Hints styles

To understand the used terminology in this sub section, please read the **Extended Window Manager Hints** section.

EWMHDonateIcon instructs fvwm to set the application ewmh icon hint with the icon that is used by fvwm if the application does not provide such hint (and if the icon used by fvwm is not an icon window). *EWMHDonateMiniIcon* does the same thing for mini icons. This allows compliant pager, taskbar, iconbox ...etc to display the same (mini) icons as fvwm. Note that on some hardware (e.g., 8−bit displays) these styles can slow down window mapping and that in general only one of these styles is needed by a compliant application. *EWMHDontDonateIcon* and *EWMHDontDonateMiniIcon* restore the defaults which are to not set any ewmh (mini) icons hints.

By default, if an application provides an ewmh icon hint of small size (i.e., height and width less than or equal to 22), then fvwm uses this icon as its mini icon. *EWMHMiniIconOverride* instructs fvwm to ignore ewmh icons and to use the mini icon provided by the *MiniIcon* style. *EWMHNoMiniIconOverride* restores the default.

EWMHUseStackingOrderHints causes fvwm to use EWMH hints and respect EWMH hints which change the window layer. *EWMHIgnoreStackingOrderHints* causes fvwm to ignore EWMH layer hints.

An application can ask for some reserved space on the desktop by a hint. In the EWMH terminology such a hint is called a strut and it is used to compute the working area and may be used for window placement and in the maximize command. *EWMHIgnoreStrutHints* causes fvwm to ignore such hints, as *EWMHUseStrutHints*, causes fvwm to use it which is the default.

EWMHIgnoreStateHints causes fvwm to ignore initial EWMH state hints when a new window is mapped. The default *EWMHUseStateHints* causes fvwm to accept such hints.

EWMHIgnoreWindowType causes fvwm to ignore EWMH window type specification. The default *!EWMHIgnoreWindowType* causes fvwm to style windows of specified types as such.

EWMHMaximizeIgnoreWorkingArea causes fvwm to ignore the EWMH working area when it executes a **Maximize** command. With *EWMHMaximizeUseWorkingArea* the EWMH working area is used as with *EWMHMaximizeUseDynamicWorkingArea* the EWMH dynamic working area is used (the default).

EWMHPlacementIgnoreWorkingArea causes fvwm to ignore the EWMH working area when it places (or places again) a window. With *EWMHPlacementUseWorkingArea* the EWMH working area is taken in account as with *EWMHPlacementUseDynamicWorkingArea* the EWMH dynamic working area is taken in account (the default). Note that with the*MinOverlapPlacement* and *MinOverlapPercentPlacement* placement policy, the way the EWMH (dynamic) working area is taken in account is configurable with the *MinOverlapPlacementPenalties* style.

Miscellaneous

The *BackingStore*, *BackingStoreOff* and *BackingStoreWindowDefault* determine if the X server uses backing store for the window or not. *BackingStore* means that the X server tries to keep the obscured parts of a window in memory. This is usually slower if the client runs on the same machine as the X server, but can be much faster if the connection is slow (see also *SaveUnder* below). *BackingStoreOff* disables backing store for the window. By default, fvwm does not enable or disable backing store itself but leaves is as the window requested it. To revert back to the application's choice, use the *BackingStoreWindowDefault* style.

Note: This style is useless if the X server does not allow backing store.

SaveUnder enables the corresponding window attribute in the X server. For a window using this

style, the X server tries to store the graphics below it in memory which is usually slower if the client runs on the same machine as the X server. *SaveUnder* may speed up fvwm if the connection to the X server is slow (e.g. over a modem link). To disable save under, use the *SaveUnderOff* style. This is the default. See also*Bac kingStore* above.

Note: This style is useless if the X server does not allow save under.

ParentalRelativity enables clients that use a background pixmap of type *ParentRelative* to achieve transparency. Fvwm modules that support transparent colorsets require this setting. *Opacity* is the default and should be used for all non−transparent clients for better performance.

MwmDecor makes fvwm attempt to recognize and respect the mwm decoration hints that applications occasionally use. To switch this style off, use the *NoDecorHint* style.

MwmFunctions makes fvwm attempt to recognize and respect the mwm prohibited operations hints that applications occasionally use. *HintOverride* makes fvwm shade out operations that mwm would prohibit, but it lets you perform the operation anyway. *NoFuncHint* allows turns off the mwm hints completely.

OLDecor makes fvwm attempt to recognize and respect the olwm and olvwm hints that many older XView and OLIT applications use. Switch this option off with *NoOLDecor*.

With *GNOMEIgnoreHints* fvwm ignores all GNOME hints for the window, even if GNOME compliance is compiled in. This is useful for those pesky applications that try to be more clever than the user and use GNOME hints to force the window manager to ignore the user's preferences. The *GNOMEUseHints* style switches back to the default behavior.

UseDecor This style is deprecated and will be removed in the future. There are plans to replace it with a more flexible solution in fvwm−3.0.

UseDecor accepts one argument: the name of a decor created with **AddToDecor**. If no decor name is specified, the "Default" decor is used. Windows do not actually contain decors, but are always assigned to one. If the decor is later modified with **AddToDecor**, the changes are visible for all windows which are assigned to it. The decor for a window can be reassigned with **ChangeDecor**.

UseStyle This style is deprecated and will be removed in the future. There are plans to replace it with a more flexible solution in fvwm−3.0.

UseStyle takes one arg, which is the name of another style. That way you can have unrelated window names easily inherit similar traits without retyping. For example:

```
Style rxvt UseStyle XTerm
```

Warning: If a style is built from one or more parent styles and the parent styles are changed, the derived style is not modified. To achieve this you have to issue the *UseStyle* line again.

Unmanaged Windows with the *Unmanaged* style option are ignored by fvwm. They are not decorated, can not be moved or resized, etc. You probably want to use **Bugopts RaiseOverUnmanaged** too. This option can be turned off with the *!Unmanaged* style. However, windows that are already ignored at the time when the option is set must be recaptured with the **Recapture** command in order to become managed.

State sets the initial value of one of the 32 user defined states which are associated with each window. The state number ranges from 0 to 31 and must be given as an argument. The states have no meaning in fvwm, but they can be checked in conditional commands like **Next** with the *State* condition and manipulated with the **State** command.

```
# turn on state 11 for xterms ...
Style xterm State 11
# ... but not for rxvts.
Style rxvt !State 11
```

Windows with the *WindowListSkip* styles do not appear in the menu that is created with the **WindowList** command or the lists shown in modules like **FvwmIconMan**. In the modules, the style can usually be ignored with an option. Please refer to the man page of the module in question for further information. To disable this feature, use the default style *WindowListHit*.

The styles *CirculateSkip* and *CirculateHit* control whether the window is considered by conditional commands, for example **Next**, *Prev* or *All*. Windows with *CirculateSkip*, are never selected by conditional commands. However, the styles can be overridden explicitly in the condition with the *CirculateHit*, *CirculateHitIcon* or *CirculateHitShaded* conditions, and some conditional commands, e.g. **Current** and *All*, do this by default. The styles*Cir culateSkipIcon*, *CirculateHitIcon*, *CirculateSkipShaded* and *CirculateHitShaded* work like *CirculateSkip* and *CirculateHit* but apply only to iconic or shaded windows. Note: if multiple ...Skip... options are combined, windows are only selected if they match none of the given conditions. So, with

```
Style * CirculateSkipIcon, CirculateSkipShaded
```

only windows that are neither iconic nor shaded are selected. Note: For historical reasons, the conditional commands understand the names of these styles as condition names. Take care not to confuse them.

Examples

```
# Change default fvwm behavior to no title−
# bars on windows! Also define a default icon.
Style *           !Title,              \
                  Icon unknown1.xpm,   \
                  BorderWidth 4,       \
                  HandleWidth 5

# now, window specific changes:
Style Fvwm*       !Handles, Sticky,    \
                  WindowListSkip,      \
                  BorderWidth 0
Style FvwmPager   StaysOnTop, BorderWidth 0
Style *lock       !Handles, Sticky,    \
                  StaysOnTop, WindowListSkip
Style xbiff       Sticky, WindowListSkip
Style FvwmButtons !Handles, Sticky,    \
                  WindowListSkip
Style sxpm        !Handles

# Put title−bars back on xterms only!
Style xterm     Title, Color black/grey

Style rxvt        Icon term.xpm
Style xterm       Icon rterm.xpm
Style xcalc       Icon xcalc.xpm
Style xbiff       Icon mail1.xpm
Style xmh         Icon mail1.xpm,      \
                  StartsOnDesk 2
Style xman        Icon xman.xpm
Style matlab      Icon math4.xpm,      \
                  StartsOnDesk 3
Style xmag        Icon magnifying_glass2.xpm
Style xgraph      Icon graphs.xpm
Style FvwmButtons Icon toolbox.xpm
```

```
Style Maker      StartsOnDesk 1
Style signal     StartsOnDesk 3

# Fire up Netscape on the second desk, in the
# middle of my 3x3 virtual desktop, and do not
# bother me with it...
Style Netscape* SkipMapping,            \
        StartsOnPage 1 1 1
```

Note that all properties for a window are or'ed together. In the above example "FvwmPager" gets the property *StaysOnTop* via an exact window name match but also gets *!Handles*, *Sticky* and *WindowListSkip* by a match to "Fvwm*". It gets *!Title* by virtue of a match to "*". If conflicting styles are specified for a window, then the last style specified is used.

WindowStyle *options*
sets attributes (styles) on the selected window. The*options* are e xactly the same as for the **Style** command.

Window Styles

AddButtonStyle button [*state*] [*style*] [−− [!]*flag* ...]
Adds a button style to *button*. *button* can be a button number, or one of "*All*", "*Left*" or "*Right*". *state* can be "*ActiveUp*", "*ActiveDown*", "*InactiveUp*" or "*InactiveDown*", or "*Active*" (the same as both "ActiveUp" and "ActiveDown") or "*Inactive*" (the same as both "InactiveUp" and "InactiveDown") or any of these 6 with "*Toggled*" prepended. The "Active" states apply to the focused window, the "Inactive" ones apply to all other windows. The "Up" states apply to the non pressed buttons, the "Down" ones apply to pressed buttons. The "Toggled" prefix refers to maximized, shaded or sticky windows that have the corresponding *MwmDecor...* button style set. Additionally, the following shortcuts may be used: "*AllNormal*", "*AllToggled*", "*AllActive*", "*AllInactive*", "*AllUp*", "*AllDown*". They are actually different masks for 4 individual states from 8 total. These are supported too: "*AllActiveUp*", "*AllActiveDown*", "*AllInactiveUp*", "*AllInactiveDown*".

If *state* is omitted, then the style is added to every state. If the *style* and *flags* are enclosed in parentheses, then multiple *state* definitions can be placed on a single line. *Flags* for additional button styles cannot be changed after definition.

Buttons are drawn in the order of definition, beginning with the most recent button style, followed by those added with **AddButtonStyle**. To clear the button style stack, change style flags, or for descriptions of available styles and flags, see the **ButtonStyle** command. Examples:

```
ButtonStyle 1 Pixmap led.xpm −− Top Left
ButtonStyle 1 ActiveDown HGradient 8 grey black
ButtonStyle All −−  UseTitleStyle
AddButtonStyle 1 \
        ActiveUp (Pixmap a.xpm) \
        ActiveDown (Pixmap b.xpm −− Top)
AddButtonStyle 1 Vector 4 50x30@1 70x70@0 30x70@0 50x30@1
```

Initially for this example all button states are set to a pixmap. The second line replaces the "ActiveDown" state with a gradient (it overrides the pixmap assigned to it in the line before, which assigned the same style to every state). Then, the *UseTitleStyle* flag is set for all buttons, which causes fvwm to draw any styles set with **TitleStyle** before drawing the buttons. Finally, **AddButtonStyle** is used to place additional pixmaps for both "ActiveUp" and "ActiveDown" states and a vector button style is drawn on top of all states.

AddTitleStyle [*state*] [*style*] [−− [!]*flag* ...]
Adds a title style to the title−bar. *state* can be "*ActiveUp*", "*ActiveDown*", "*InactiveUp*" or "*InactiveDown*", or "*Active*" (the same as both "ActiveUp" and "ActiveDown") or "*Inactive*" (the same as both "InactiveUp" and "InactiveDown") or any of these 6 with "Toggled" prepended. If

state is omitted, then the style is added to every state. If the *style* and *flags* are enclosed in parentheses, then multiple *state* definitions can be placed on a single line. This command is quite similar to the **AddButtonStyle** command.

Title−bars are drawn in the order of definition, beginning with the most recent **TitleStyle**, followed by those added with **AddTitleStyle**. To clear the title style stack, change style flags, or for the descriptions of available styles and flags, see the **TitleStyle** and **ButtonStyle** commands.

AddToDecor *decor*

This command is deprecated and will be removed in the future. There are plans to replace it with a more flexible solution in fvwm−3.0.

Add or divert commands to the decor named *decor*. A decor is a name given to the set of commands which affect button styles, title−bar styles and border styles. If *decor* does not exist it is created; otherwise the existing *decor* is modified. Note: Earlier versions allowed to use the **HilightColor**, **HilightColorset** and **WindowFont** commands in decors. This is no longer possible. Please use the**Style** command with the *Hilight...* and *F ont* options.

New decors start out exactly like the "default" decor without any style definitions. A given decor may be applied to a set of windows with the *UseDecor* option of the **Style** command. Modifying an existing decor affects all windows which are currently assigned to it.

AddToDecor is similar in usage to the **AddToMenu** and **AddToFunc** commands, except that menus and functions are replaced by **ButtonStyle**, **AddButtonStyle**, **TitleStyle**, **AddTitleStyle** and **BorderStyle** commands. Decors created with **AddToDecor** can be manipulated with **ChangeDecor**, **DestroyDecor**, **UpdateDecor** and the **Style** option.

The following example creates a decor "FlatDecor" and style "FlatStyle". They are distinct entities:

```
AddToDecor FlatDecor
+ ButtonStyle All Active (−− flat) Inactive (−− flat)
+ TitleStyle  −− flat
+ BorderStyle −− HiddenHandles NoInset

Style FlatStyle \
        UseDecor FlatDecor, HandleWidth 4, ForeColor white, \
        BackColor grey40, HilightFore black, HilightBack grey70

Style xterm UseStyle FlatStyle
```

An existing window's decor may be reassigned with **ChangeDecor**. A decor can be destroyed with **DestroyDecor**.

```
DestroyDecor FlatDecor
AddToDecor FlatDecor ...

Style FlatStyle UseDecor FlatDecor
```

and now apply the style again:

```
Style xterm UseStyle FlatStyle
```

BorderStyle *state* [*style*] [−− [!]*flag* ...]

Defines a border style for windows. *state* can be either "*Active*" or "*Inactive*". If*state* is omitted, then the style is set for both states. If the *style* and *flags* are enclosed in parentheses, then multiple *state* definitions can be specified per line.

style is a subset of the available button styles, and can only be *TiledPixmap* (uniform pixmaps which match the bevel colors work best this way) or *Colorset*. If a '!' is prefixed to any *flag*, the

behavior is negated. If*style* is not specified, then one can change flags without resetting the style.

The *HiddenHandles* flag hides the corner handle dividing lines on windows with handles (this option has no effect for !*Handles* windows). By default, *HiddenHandles* is disabled.

The *NoInset* flag supplements *HiddenHandles*. If given, the inner bevel around the window frame is not drawn. If*HiddenHandles* is not specified, the frame looks a little strange.

Raised causes a raised relief pattern to be drawn (default). *Sunk* causes a sunken relief pattern to be drawn. *Flat* inhibits the relief pattern from being drawn.

To decorate the active and inactive window borders with a textured pixmap, one might specify:

```
BorderStyle Active TiledPixmap marble.xpm
BorderStyle Inactive TiledPixmap granite.xpm
BorderStyle Active -- HiddenHandles NoInset
```

To clear the style for both states:

```
BorderStyle Simple
```

To clear for a single state:

```
BorderStyle Active Simple
```

To unset a flag for a given state:

```
BorderStyle Inactive -- !NoInset
```

title−bar buttons can inherit the border style with the *UseBorderStyle* flag (see **ButtonStyle**).

ButtonState [ActiveDown *bool*] [Inactive *bool*] [InactiveDown *bool*]
The **ButtonState** command controls which states of the window titles and title buttons are used. The default is to use all four states: "ActiveUp", "ActiveDown", "InactiveUp" and "InactiveDown" (see **ButtonStyle** and **TitleStyle** commands). The *bool* argument after the key word controls if the designated state is used ("True") or not ("False"). The*bool* flag is the same as other commands, and not limited to just "True" or "False"; "Yes" and "No" may also be used. The "ActiveUp" state cannot be deactivated. If no arguments are provided or the given arguments are illegal, the default is restored.

If *ActiveDown* argument is "False", no different button style for the pressed down buttons used, instead "ActiveUp" state is used even when button is pressed.

If *Inactive* argument is "False", focused and unfocused windows look similarly, the corresponding "Active" states are always used.

If *InactiveDown* argument is "False" (only applied when *Inactive* is "True"), the pressed titles and title buttons in non−focused windows are drawn using "InactiveUp" or "ActiveUp" states depending on the values of the other key words.

ButtonStyle button [*state*] [*style*] [−− [!]*flag* ...]
Sets the button style for a title−bar button. *button* is the title−bar button number between 0 and 9, or one of "*All*", "*Left*", "*Right*", or "*Reset*". Button numbering is described in the**Mouse** command section. If the*style* and *fla gs* are enclosed in parentheses, then multiple *state* definitions can be specified per line.

state refers to which button state should be set. Button states are defined as follows: "*ActiveUp*" and "*ActiveDown*" refer to the un−pressed and pressed states for buttons on active windows; while the "*InactiveUp*" and "*InactiveDown*" states denote buttons on inactive windows. The shortcut "*Active*" denotes both "ActiveUp" and "ActiveDown" states. Shortcut "*Inactive*" denotes both "InactiveUp" and "InactiveDown" states. The similar state names like just described, but with the "Toggled" prefix are used instead for title buttons which have one of the *MwmDecorMax*, *MwmDecorShade*, *MwmDecorStick* or *MwmDecorLayer* hints, if the window is maximized,

shaded, sticky or placed on specific layer, respectively.

```
AddToDecor Default
+ ButtonStyle 6                \
  Vector 4 50x25@1 85x75@0 15x75@0 50x25@1
+ ButtonStyle 6 ToggledActiveUp   \
  Vector 4 50x75@0 85x25@1 15x25@0 50x75@0
+ ButtonStyle 6 ToggledActiveDown \
  Vector 4 50x75@0 85x25@1 15x25@0 50x75@0
+ ButtonStyle 6 ToggledInactive   \
  Vector 4 50x75@0 85x25@1 15x25@0 50x75@0
+ ButtonStyle 6 − MwmDecorShade
Mouse 0 6 N WindowShade
```

Additionally, the following shortcuts may be used: "*AllNormal*", "*AllToggled*", "*AllActive*", "*AllInactive*", "*AllUp*", "*AllDown*". They are actually different masks for 4 individual states from 8 total. These are supported too: "*AllActiveUp*", "*AllActiveDown*", "*AllInactiveUp*", "*AllInactiveDown*".

If *state* is specified, that particular button state is set. If *state* is omitted, every state is set. Specifying a style destroys the current style (use **AddButtonStyle** to avoid this).

If *style* is omitted, then state−dependent flags can be set for the primary button style without destroying the current style. Examples (each line should be considered independent):

```
ButtonStyle Left −− flat
ButtonStyle All ActiveUp (−− flat) Inactive (−− flat)
```

The first line sets every state of the left buttons to flat, while the second sets only the "ActiveUp" and "Inactive" states of every button to flat (only flags are changed; the buttons' individual styles are not changed).

If you want to reset all buttons to their defaults:

```
ButtonStyle Reset
```

To reset the "ActiveUp" button state of button 1 to the default:

```
ButtonStyle 1 ActiveUp Default
```

To reset all button states of button 1 to the default of button number 2:

```
ButtonStyle 1 Default 2
```

For any button, multiple *state* definitions can be given on one line by enclosing the *style* and *flags* in parentheses. If only one definition per line is given the parentheses can be omitted.

flags affect the specified *state*. If a '!' is prefixed to any *flag*, its behavior is negated. The available state−dependent flags for all styles are described here (the **ButtonStyle** entry deals with state−independent flags).

Raised causes a raised relief pattern to be drawn.

Sunk causes a sunken relief pattern to be drawn.

Flat inhibits the relief pattern from being drawn.

UseTitleStyle causes the given button state to render the current title style before rendering the buttons' own styles. The *Raised*, *Flat* and *Sunk* **TitleStyle** flags are ignored since they are redundant in this context.

UseBorderStyle causes the button to inherit the decorated **BorderStyle** options.

Raised, *Sunk* and *Flat* are mutually exclusive, and can be specified for the initial **ButtonStyle** only. *UseTitleStyle* and *UseBorderStyle* are also mutually exclusive (both can be off however). The default is *Raised* with both *UseBorderStyle and UseTitleStyle* left unset.

Important
for the "ActiveDown" and "InactiveDown" states: When a button is pressed, the relief is inverted. Because of this, to obtain the raised look in "ActiveDown" or "InactiveDown" states you must specify the opposite of the desired relief (i.e. *Sunk* for "ActiveDown" or "InactiveDown"). This behavior is consistent, but may seem confusing at first. The same applies to the "Toggled" states.

Button styles are classified as non−destructive, partially destructive, or fully destructive. Non−destructive styles do not affect the image. Partially destructive styles can obscure some or all parts of the underlying image (i.e. *Pixmap*). Fully destructive styles obscure the entire underlying image (i.e. *Solid* or one of the *gradient* styles). Thus, if stacking styles with **AddButtonStyle** (or **AddTitleStyle** for title−bars), use care in sequencing styles to minimize redraw.

The available styles are:

Simple, *Default*, *Solid*, *Colorset*, *Vector*, *?Gradient*, *Pixmap*, *AdjustedPixmap*, *ShrunkPixmap*, *StretchedPixmap*, *TiledPixmap*, *MiniIcon*

The description of these styles and their arguments follow:

The *Simple* style does nothing. There are no arguments, and this style is an example of a non−destructive button style.

The *Default* style conditionally accepts one argument: a number which specifies the default button number to load. If the style command given is **ButtonStyle** or **AddButtonStyle**, the argument is optional (if given, it overrides the current button). If a command other than**ButtonStyle** or **AddButtonStyle** is used, the number must be specified.

The *Solid* style fills the button with a solid color. The relief border color is not affected. The color is specified as a single argument. This style is fully destructive.

The *Colorset cs* [*alpha*] style fills the button with the Colorset *cs*. The optional*alpha* ar gument is a percentage between 0 and 100. It causes fvwm to merge the colorset background onto the button using this percentage. If the percentage is 0 the colorset background is hidden and if it is 100 the colorset background is fully applied. The default is 100. So, the destructiveness depends on the *alpha* argument.

The *Vector num X***[***offset***p]x***Y***[***offset***p]@***C* **...** style draws a line pattern. Since this is a standard button style, the keyword *Vector* is optional, *num* is a number of point specifications of the form *X***[***offset***p]x***Y***[***offset***p]@***C* **...** *X* and *Y* are point coordinates inside the button, given in percents (from 0 to 100). An optional absolute *offset* in pixels, can be given as "+<offset>p" for a positive or "−<offset>p" for a negative offset.

C specifies a line color (0 − the shadow color, 1 − the highlight color, 2 − the background color, 3 − the foreground color, 4 − only move the point, do not draw). The first point color is not used. You can use up to 10000 points in a line pattern. This style is partially destructive.

The specification is a little cumbersome:

```
ButtonStyle 2 Vector 4 50x30@1 70x70@0 30x70@0 50x30@1
```

then the button 2 decoration uses a 4−point pattern consisting of a line from (x=50,y=30) to (70,70) in the shadow color (@0), and then to (30,70) in the shadow color, and finally to (50,30) in the highlight color (@1). Is that too confusing? See the fvwm web pages for some examples with screenshots.

A more complex example of *Vector*:

```
ButtonStyle 8 Vector 10 45x65@2 45x75@3 \
```

```
  20x75@3 20x50@3 35x50@3 35x65@1 35x25@1 \
  75x25@1 75x65@0 35x65@0
ButtonStyle 0 Vector 10 45x65@2 45x75@0 \
  20x75@0 20x50@1 45x50@1 45x65@0 75x65@3 \
  75x25@3 35x25@3 35x47@3
```

The *?Gradient* styles denote color gradients. Fill in the question mark with any one of the defined gradient types. Please refer to the **Color Gradients** section for a description of the gradient syntax. The gradient styles are fully destructive.

The *Pixmap* style displays a pixmap. A pixmap should be specified as an argument. For example, the following would give button number 2 the same pixmap for all 4 states (2 active and 2 inactive), and button number 4 all different pixmaps.

```
ButtonStyle 2 Pixmap my_pixmap.xpm
ButtonStyle 4 \
        ActiveUp (Pixmap activeup.xpm) \
        ActiveDown (Pixmap activedown.xpm) \
        Inactive (Pixmap inactiveup.xpm)
ButtonStyle 4 \
        InactiveDown Pixmap inactivedown.xpm
```

The pixmap specification can be given as an absolute or relative pathname (see **ImagePath**). If the pixmap cannot be found, the button style reverts to *Simple*. Flags specific to the*Pixmap* style are *Left, Right, Top*, and *Bottom*. These can be used to justify the pixmap (default is centered for both directions). Pixmap transparency is used for the color "None." This style is partially destructive.

The *AdjustedPixmap* style is similar to the *Pixmap* style. But the image is resized to exactly fit the button.

The *ShrunkPixmap* style is similar to the *Pixmap* style. But if the image is bigger than the button the image is resized to fit into the button.

The *StretchedPixmap* style is similar to the *Pixmap* style. But if the image is smaller than the button the image is resized to cover the button.

The *TiledPixmap* style accepts a pixmap to be tiled as the button background. One pixmap is specified as an argument. Pixmap transparency is not used. This style is fully destructive.

The *MiniIcon* style draws the window's miniature icon in the button, which is specified with the *MiniIcon* option of the **Style** command. This button style accepts no arguments. Example:

```
Style * MiniIcon mini-bx2.xpm
Style xterm MiniIcon mini-term.xpm
Style Emacs MiniIcon mini-doc.xpm

ButtonStyle 1 MiniIcon
```

ButtonStyle *button* - [!]*flag ...*

Sets state-independent flags for the specified *button*. State-independent flags affect button behavior. Each*fla g* is separated by a space. If a '!' is prefixed to the flag then the behavior is negated. The special flag*Clear* clears an y existing flags.

The following flags are usually used to tell fvwm which buttons should be affected by mwm function hints (see *MwmFunctions* option of the **Style** command. This is not done automatically since you might have buttons bound to complex functions, for instance.

MwmDecorMenu should be assigned to title-bar buttons which display a menu. The default assignment is the leftmost button. When a window with the *MwmFunctions* **Style** option requests not to show this button, it is hidden.

MwmDecorMin should be assigned to title−bar buttons which minimize or iconify the window. The default assignment is the second button over from the rightmost button. When a window with the *MwmFunctions* **Style** option requests not to show this button, it is hidden.

MwmDecorMax should be assigned to title−bar buttons which maximize the window. The default assignment is the rightmost button. When a window with the *MwmFunctions* **Style** option requests not to show this button, it is hidden. When the window is maximized, the vector pattern on the button looks pressed in.

MwmDecorShade should be assigned to title−bar buttons which shade the window (see **WindowShade** command). When the window is shaded, the vector pattern on the button looks pressed in.

MwmDecorStick should be assigned to title−bar buttons which make the window sticky. When the window is sticky, the vector pattern on the button looks pressed in.

The flag *MwmDecorLayer layer* should be assigned to title−bar buttons which place the window in the layer numbered *layer*. When the window is on that specific layer, the vector pattern on the button looks pressed in.

ChangeDecor *decor*

This command is deprecated and will be removed in the future. There are plans to replace it with a more flexible solution in fvwm−3.0.

Changes the decor of a window to *decor*. *decor* is "Default" or the name of a decor defined with **AddToDecor**. If*decor* is in valid, nothing occurs. If called from somewhere in a window or its border, then that window is affected. If called from the root window the user is allowed to select the target window. **ChangeDecor** only affects attributes which can be set using the **AddToDecor** command.

```
ChangeDecor CustomDecor1
```

DestroyDecor [recreate] *decor*

This command is deprecated and will be removed in the future. There are plans to replace it with a more flexible solution in fvwm−3.0.

Deletes the *decor* defined with **AddToDecor**, so that subsequent references to it are no longer valid. Windows using this *decor* revert to the "Default" decor. The optional parameter*r ecreate* tells fvwm not to throw away the decor completely but to throw away only its contents. If the decor is created again later, windows do not use it before the *UseDecor* style is applied again unless the decor was destroyed with the *recreate* option. The decor named "Default" cannot be destroyed.

```
DestroyDecor CustomDecor1
```

TitleStyle [*justification*] [Height [*num*]] [MinHeight [*num*]]

Sets attributes for the title−bar. Justifications can be*Center ed*, *RightJustified* or *LeftJustified*. *Height* sets the title bar's height to an amount in pixels. *MinHeight* sets the minimal height in pixels of the title bar. Defaults are *Centered*, the window's font height and no minimal height. To reset the font height to the default value, omit the *num* argument after the *Height* keyword. The *MinHeight* height is reset by *Height* or if given with no argument. Example:

```
TitleStyle LeftJustified Height 24
```

TitleStyle [*state*] [*style*] [−− [!]*flag* ...]

Sets the style for the title−bar. See also**AddT itleStyle** and **ButtonStyle** *state* can be one of "*ActiveUp*", "*ActiveDown*", "*InactiveUp*", or "*InactiveDown*". Shortcuts like "*Active*" and "*Inactive*" are allowed. The states with the "Toggled" prefix are allowed too, the title itself does not use "Toggled" states, but these states are used for the buttons with **ButtonStyle** *UseTitleStyle*. If *state* is omitted, then the *style* is added to every state. If parentheses are placed around the *style*

and *flags*, then multiple state definitions can be given per line. *style* can be omitted so that flags can be set while not destroying the current style.

If a '!' is prefixed to any *flag*, its behavior is negated. Valid flags for each state include *Raised*, *Flat* and *Sunk* (these are mutually exclusive). The default is *Raised*. See the note in**ButtonStyle** regarding the "*ActiveDown*" state. Examples:

```
TitleStyle ActiveUp HGradient 16 navy black
TitleStyle \
        ActiveDown (Solid red -- flat) \
        Inactive (TiledPixmap wood.xpm)
TitleStyle \
        ActiveUp (-- Flat) \
        ActiveDown (-- Raised) \
        InactiveUp (-- Flat) \
        InactiveDown (-- Sunk)
```

This sets the "ActiveUp" state to a horizontal gradient, the "ActiveDown" state to solid red, and the "Inactive" states to a tiled wood pixmap. Finally, "ActiveUp" and "InactiveUp" are set to look flat, while "ActiveDown" set to be sunk (the *Raised* flag for the "ActiveDown" state causes it to appear sunk due to relief inversion), and "InactiveDown" is set to look raised. An example which sets flags for all states:

```
TitleStyle -- flat
```

For a flattened look:

```
TitleStyle -- flat
ButtonStyle All Active (-- flat) Inactive (-- flat)
```

TitleStyle accepts all the **ButtonStyle** styles and arguments:

Simple, *Default*, *Solid*, *Colorset*, *Vector*, *?Gradient*, *Pixmap*, *AdjustedPixmap*, *ShrunkPixmap*, *StretchedPixmap*, *TiledPixmap*, *MiniIcon*.

See the **ButtonStyle** command for a description of all these styles and their arguments.

In addition to these styles **TitleStyle** accepts a powerful *MultiPixmap* option. This allows you to specify different pixmaps, colorsets or colors for different parts of the titlebar. Some of them are tiled or stretched to fit a particular space; others are discrete "transition" images. The definable *sections* are:

Main
: The full titlebar

LeftMain
: Left of title text

RightMain
: Right of title text

UnderText
: Underneath title text

LeftOfText
: just to the left of the title text

RightOfText
: just to the right of the title text

LeftEnd
: at the far left end of the titlebar (just after left buttons if any)

RightEnd
at the far right end of the titlebar (just before right buttons if any)

Buttons
under buttons in case of *UseTitleStyle*

LeftButtons
under left buttons in case of *UseTitleStyle*

RightButtons
under right buttons in case of *UseTitleStyle*

None of these are mandatory except for *Main* (or, if you do not define *Main* you must define both *LeftMain* and *RightMain*). If no*Buttons* pixmaps are defined and *UseT itleStyle* is specified for one or more buttons, *Main*, *LeftMain* or *RightMain* are used as appropriate.

The syntax for this style type is:

MultiPixmap section style arg, ...

continuing for whatever you want to define. The *style* can be either *TiledPixmap*, *AdjustedPixmap*, *Colorset* or *Solid*. See the**ButtonStyle** command for the description of these styles. In the case of a transition section, *LeftEnd*, *LeftOfText*, *RightOfText* or *RightEnd*, *AdjustedPixmap* only resize the pixmap in the "y" direction. For the *Colorset* and *Solid* styles a width of the half of the title bar height is assumed for the transition sections.

An example:

```
MultiPixmap Main AdjustedPixmap foo.xpm, \
    UnderText TiledPixmap bar.xpm, \
    Buttons Colorset 2
```

Note that the old syntax is still supported: if the style is omitted, *TiledPixmap* is assumed and adding "(stretched)" between the section and the file name implies *AdjustedPixmap*.

UpdateDecor [*decor*]
This command is deprecated and will be removed in the future. There are plans to replace it with a more flexible solution in fvwm−3.0.

This command is kept mainly for backward compatibility. Since all elements of a decor are updated immediately when they are changed, this command is mostly useless.

Updates window decorations. *decor* is an optional argument which specifies the *decor* to update. If given, only windows which are assigned to that particular *decor* are updated. This command is useful, for instance, after a **ButtonStyle**, **TitleStyle** or **BorderStyle** (possibly used in conjunction with **AddToDecor**). Specifying an invalid decor results in all windows being updated. This command is less disturbing than **Recapture**, but does not affect window style options as **Recapture** does.

Controlling the Virtual Desktop

Desk *arg1* [*arg2*] [*min max*]
This command has been renamed. Please see **GotoDesk** command.

DesktopName *desk name*
Defines the name of the desktop number *desk* to *name*. This name is used in the**W indowList** command and in the **FvwmPager** where it override the *Label* configuration option. Moreover, if consecutive names starting from desktop 0 are defined, then these names can be used by any EWMH compliant application (as a pager).

DesktopSize *Horizontal*x*Vertical*
Defines the virtual desktop size in units of the physical screen size.

EdgeResistance *delay***EdgeResistance** *scrolling moving* [*xinerama−scrolling*]
Tells how hard it should be to change the desktop viewport by moving the mouse over the edge of the screen. The parameter tells how many milliseconds the pointer must spend on the screen edge before fvwm moves the viewport. This is intended for people who use

EdgeScroll 100 100

but find themselves accidentally flipping pages when they do not want to. If −1 is given as the delay, scrolling is disabled completely.

The second form of invocation with two or three arguments is obsolete and should be replaced with the following three commands as needed:

EdgeResistance *scrolling*
Style * EdgeMoveDelay *scrolling*
Style * EdgeMoveResistance *moving*
or
Style * EdgeMoveResistance *moving xinerama−scrolling*

Fvwm does this substitution automatically and prints a warning.

EdgeScroll *horizontal*[p] *vertical*[p] [wrap | wrapx | wrapy]
Specifies the percentage of a page to scroll when the cursor hits the edge of a page. A trailing '*p*' changes the interpretation to mean pixels. If you do not want any paging or scrolling when you hit the edge of a page include

EdgeScroll 0 0

in your *config* file, or possibly better, set the **EdgeThickness** to zero. See the **EdgeThickness** command. If you want whole pages, use

EdgeScroll 100 100

Both *horizontal* and *vertical* should be positive numbers.

If the *horizontal* and *vertical* percentages are multiplied by 1000 or one of the keywords *wrap*, *wrapx* and *wrapy* is given then scrolling wraps around at the edge of the desktop. If

EdgeScroll 100000 100000

is used fvwm scrolls by whole pages, wrapping around at the edge of the desktop.

EdgeThickness 0 | 1 | 2
This is the width or height of the invisible window that fvwm creates on the edges of the screen that are used for the edge scrolling feature.

In order to enable page scrolling via the mouse, four windows called the "pan frames" are placed at the very edge of the screen. This is how fvwm detects the mouse's presence at the window edge. Because of the way this works, they need to be at the top of the stack and eat mouse events, so if you have any kind of error along the lines of: "mouse clicks at the edge of the screen do the wrong thing" you're having trouble with the pan frames and (assuming you do not use the mouse to flip between pages) should set the EdgeThickness to 0.

A value of 0 completely disables mouse edge scrolling, even while dragging a window. 1 gives the smallest pan frames, which seem to work best except on some servers.

2 is the default.

Pan frames of 1 or 2 pixels can sometimes be confusing, for example, if you drag a window over the edge of the screen, so that it straddles a pan frame, clicks on the window, near the edge of the screen are treated as clicks on the root window.

EwmhBaseStruts *left right top bottom*

Where left, right, top and bottom are positive or null integers which define bands at the edge of the screen. *left* defines a band on the left of your screen of width *left*, *right* defines a band on the right of your screen of width *right*, *top* defines a band on the top of your screen of height *top* and *bottom* defines a band on the bottom of your screen of height *bottom*. The unit is the pixel and the default is 0 0 0 0. These areas define additional reserved space to the reserved space defined by some ewmh compliant applications. This is used to compute the Working Area. See the **Extended Window Manager Hints** section for a definition of the Working Area.

EwmhNumberOfDesktops *num* [*max*]

This command is useful only for an ewmh compliant pager or taskbar (as kpager or kicker taskbar) and not for fvwm modules (**FvwmPager** or **FvwmIconMan**). It causes a compliant application to consider at least *num* desktops (desktop 0 to desktop *num*−1). The optional argument *max* causes a compliant application to never consider more than *max* desktops. If *max* is 0 (the default) there is no limitation. The actual number of desktops is determined dynamically. It is at least *num*, but it can be d if there is a window on desktop d−1 (or if the current desktop is desktop d−1) and d is less or equal to *max* or *max* is null. Moreover, a compliant pager can ask to change *num* itself. This is accepted by fvwm only if this number is less than or equal to*max* or if *max* is null. Note that negative desktops are not supported by the ewmh specification. The default is 4 0.

GotoDesk [prev | *arg1* [*arg2*] [*min max*]]

Switches the current viewport to another desktop (workspace, room).

The command takes 1, 2, 3, or 4 arguments. A single argument is interpreted as a relative desk number. Two arguments are understood as a relative and an absolute desk number. Three arguments specify a relative desk and the minimum and maximum of the allowable range. Four arguments specify the relative, absolute, minimum and maximum values. (Desktop numbers can be negative). If a literal*pr ev* is given as the single argument, the last visited desk number is used.

If *arg1* is non zero then the next desktop number is the current desktop number plus *arg1*.

If *arg1* is zero then the new desktop number is *arg2*. (If*ar g2* is not present, then the command has no effect.)

If *min* and *max* are given, the new desktop number is no smaller than *min* and no bigger than *max*. Values out of this range are truncated (if you gave an absolute desk number) or wrapped around (if you gave a relative desk number).

The syntax is the same as for **MoveToDesk**, which moves a window to a different desktop.

The number of active desktops is determined dynamically. Only desktops which contain windows or are currently being displayed are active. Desktop numbers must be between 2147483647 and −2147483648 (is that enough?).

GotoDeskAndPage prev | *desk xpage ypage*

Switches the current viewport to another desktop and page, similar to the **GotoDesk** and **GotoPage** commands. The new desk is *desk* and the new page is (*xpage*,*ypage*).

GotoPage prev | [*options*] *x*[p] *y*[p]

Moves the desktop viewport to page (x,y). The upper left page is (0,0), the upper right is (M,0), where M is one less than the current number of horizontal pages specified in the **DesktopSize** command. The lower left page is (0,N), and the lower right page is (M,N), where N is the desktop's vertical size as specified in the **DesktopSize** command. To switch to a page relative to the current one add a trailing '*p*' after any or both numerical arguments.

Possible *options* are *wrapx* and *wrapy* to wrap around the x or y coordinate when the viewport is moved beyond the border of the desktop.

To go to the last visited page use *prev* as the first argument. The**GotoP age** function should not be used in a pop−up menu.

Examples:

```
# Go to page (2,3)
GotoPage 2 3

# Go to lowest and rightmost page
GotoPage −1 −1

# Go to last page visited
GotoPage prev

# Go two pages to the right and one page up
GotoPage +2p −1p
```

Scroll [*horizonal*[p] *vertical*[p] | reverse]

Scrolls the virtual desktop's viewport by *horizontal* pages in the x−direction and *vertical* pages in the y−direction or starts interactive scrolling of the viewport. Either or both entries may be negative. Both*horizontal* and *vertical* v alues are expressed in percent of pages, so

```
Scroll 100 100
```

means to scroll down and right by one full page.

```
Scroll 50 25
```

means to scroll right half a page and down a quarter of a page. The **Scroll** function should not be called from pop−up menus. Normally, scrolling stops at the edge of the desktop.

If the *horizontal* and *vertical* percentages are 100 or more and are multiplied by 1000 then scrolling wraps around at the edge of the desktop. If

```
Scroll 100000 0
```

is executed over and over fvwm moves to the next desktop page on each execution and wraps around at the edge of the desktop, so that every page is hit in turn.

If the letter '*p*' is appended to each coordinate (*horizontal* and/or *vertical*), then the scroll amount is measured in pixels.

Without arguments or if the option *reverse* is given interactive scrolling takes place. The viewport scrolls as the mouse is moved. With the *reverse* option scrolling is done in opposite direction of the mouse movement, and without it scrolling in the same direction as the mouse.

The binding

```
Mouse 1 A CM Scroll reverse
```

gives an effect of grabbing and dragging the viewport with button 1 if Control and Meta is pressed.

Xinerama [bool]

Enables Xinerama support if the boolean argument is true and disables it if the argument is false. Calling this command without arguments turns on Xinerama support if it was disabled before and turns it off if it was enabled. For example:

```
# Turn Xinerama support on, use primary screen 2
XineramaPrimaryScreen 2
Xinerama on
# Turn it off again
Xinerama off
```

XineramaPrimaryScreen [primary−screen]

Takes an integer number or 'g' or 'c' as its argument. A number is taken as the number of the Xinerama screen that is to be used as the primary screen. The primary screen can be used as the

preferred screen to place windows with

XineramaPrimaryScreen <screen number>
Style * StartsOnScreen p

The primary screen is used in some of the modules and for the default icon box too. Any number that is zero or more is taken as the primary screen's number. Instead, the letter 'c' indicates to use the current screen (containing the pointer) whenever the primary screen is used. This may be very confusing under some circumstances. With 'g', the global screen is used as the primary screen, effectively disabling the primary screen. Calling this function with any other argument (including none) resets the primary screen to 0.

XineramaSls [bool]
For multi−screen implementations other than Xinerama, such as Single Logical Screen, it is possible to simulate a Xinerama configuration if the total screen seen by fvwm is made up of equal sized monitors in a rectangular grid. The **XineramaSls** command turns SLS support on or off or toggles it to the opposite state, depending on if the boolean argument is "True", "False" or "toggle". If no argument is given, this is treated like "toggle". The default layout uses one by one screens. To configure the layout, use the **XineramaSlsSize** or **XineramaSlsScreens** command.

XineramaSlsSize *Horizontal Vertical*
This command configures the layout of the Single Logical screen feature. It takes two arguments, *Horizontal* and *Vertical* which must be an integer value dividing evenly into the total desktop width, and height. For an example with two monitors side by side which appear as one screen through the X−Server with the right screen as the primary screen, use:

XineramaSlsSize 2x1
XineramaSls On
XineramaPrimaryScreen 1
Xinerama On

XineramaSlsScreens number−of−screens [*screen−spec* ...]
This command configures the layout of the Single Logical screen feature. Its first argument is the number of screens to use. It must be followed by exactly this number of *screen−spec* arguments. Each of these can be written either in standard X geometry format: "<width>x<height>+<x>+<y>" or as a space separated list of numbers: "x y width height". Both ways of describing screens can be mixed in a single command. All four numbers must be supplied. The*x* and *y* v alues specify the origin of the screen in relation to the global screen's origin while *width* and *height* specify the size of the screen in pixels. No checks are done if the geometries make sense, so it is possible to define overlapping screens (with random results) or screens that are not visible at all.

XineramaSlsScreens 3 \
512x768+0+0 512x300+512+0 512 300 512 468
XineramaSls On
XineramaPrimaryScreen 1
Xinerama On

User Functions and Shell Commands

AddToFunc [*name* [I | J | M | C | H | D *action*]]
Begins or adds to a function definition. Here is an example:

AddToFunc Move−or−Raise I **Raise**
+ M **Move**
+ D **Lower**

The function name is "Move−or−Raise", and it could be invoked from a menu or a mouse binding or key binding:

Mouse 1 TS A Move−or−Raise

The *name* must not contain embedded whitespace. No guarantees are made whether function names with embedded whitespace work or not. This behavior may also change in the future without further notice. The letter before the *action* tells what kind of action triggers the command which follows it. *'I'* stands for "Immediate", and is executed as soon as the function is invoked. *'J'* is similar to "Immediate" but is delayed until a button is pressed or released or the pointer is moved, or the function completes. It is always executed before the other function actions. *'M'* stands for "Motion", i.e. if the user starts moving the mouse. *'C'* stands for "Click", i.e., if the user presses and releases the mouse button. *'H'* stands for "Hold", i.e. if the user presses a mouse button and holds it down for more than **ClickTime** milliseconds. *'D'* stands for "Double−click". The action *'I'* causes an action to be performed on the button−press, if the function is invoked with prior knowledge of which window to act on.

There is a number of predefined symbols that are replaced by certain values if they appear on the command line. Please refer to the **Command Expansion** section for details.

Warning
Please read the comments on executing complex functions in the section **Scripting and Complex Functions**.

Examples:

If you call

Key F10 R A **Function** MailFunction xmh "−font fixed"

and "MailFunction" is

AddToFunc MailFunction
+ I **Next** ($0) **Iconify** off
+ I **Next** (AcceptsFocus, $0) **Focus**
+ I **None** ($0) **Exec** exec $0 $1

Then the last line of the function becomes

+ I **None** (xmh) **Exec** exec xmh −font fixed

The expansion is performed as the function is executed, so you can use the same function with all sorts of different arguments. You could use

Key F11 R A **Function** MailFunction zmail "−bg pink"

in the same *config*, if you wanted. An example of using "$[w.id]" is:

AddToFunc PrintFunction
+ I **Raise**
+ I **Exec** xdpr −id $[w.id]

Note that "$$" is expanded to '$'.

Another example: bind right mouse button within the window button number 6 (this is a minimize button for the win95 theme) to iconify all windows of the same resource:

AddToFunc FuncIconifySameResource "I" **All** ($0) **Iconify** on
Mouse 3 6 A FuncIconifySameResource $[w.resource]

Beep

As might be expected, this makes the terminal beep.

DestroyFunc *function*
Deletes a function, so that subsequent references to it are no longer valid. You can use this to change the contents of a function during a fvwm session. The function can be rebuilt using **AddToFunc**.

DestroyFunc PrintFunction

Echo *string*
Prints a message to *stderr*. Potentially useful for debugging things in your *config*.

Echo Beginning style definitions...

EchoFuncDefinition *function*
The **EchoFuncDefinition** is similar to the **Echo** command but prints the definition for the given *function* to *stderr*. It is useful to find out how fvwm handles quoting and for debugging functions

Exec *command*
Executes *command*. You should not use an ampersand '&' at the end of the command. You probably want to use an additional "exec" at the beginning of *command*. Without that, the shell that fvwm invokes to run your command stays until the command exits. In effect, you'll have twice as many processes running as you need. Note that some shells are smart enough to avoid this, but it never hurts to include the "exec" anyway.

The following example binds function key F1 in the root window, with no modifiers, to the exec function. The program rxvt is started with an assortment of options.

Key F1 R N Exec exec rxvt −fg yellow −bg blue \
−e /bin/tcsh

Note that this function doesn't wait for *command* to complete, so things like:

Exec "echo **AddToMenu** ... > /tmp/file"
Read /tmp/file

do not work reliably (see the **PipeRead** command).

ExecUseShell [*shell*]
Makes the **Exec** command use the specified shell, or the value of the *$SHELL* environment variable if no shell is specified, instead of the default Bourne shell (*/bin/sh*).

ExecUseShell
ExecUseShell /usr/local/bin/tcsh

Function *FunctionName*
Used to bind a previously defined function to a key or mouse button. The following example binds mouse button 1 to a function called "Move−or−Raise", whose definition was provided as an example earlier in this man page. After performing this binding fvwm executes the "move−or−raise" function whenever button 1 is pressed in a window's title−bar.

Mouse 1 T A Function Move−or−Raise

The keyword **Function** may be omitted if *FunctionName* does not coincide with an fvwm command.

Warning: Please read the comments on executing complex functions in the section **Scripting and Complex Functions**.

InfoStoreAdd *key value*
Stores the *value* at the given *key*. This is useful to store generic information used in the lifetime of an fvwm config file. For example storing program preferences for opening video files.

The purpose of this command is to store internal information to fvwm which can be used bu fvwm

functions, or when opening programs of a certain type. Previous to this command the only way to do this was via **SetEnv** but this is discouraged because it places such information in the environment, which pollutes it and makes the information global to other processes started by fvwm which may then modify them which might not be what's wanted. Hence the point of **InfoStoreAdd** is to still allow for such information to be stored, but kept internal to fvwm.

In this way, one can build up as many key/value pairs as needed. Recalling the value of a given key happens through fvwm's usual expansion mechanism. See the **Command Expansion** section for more details. For example:

```
InfoStoreAdd teddybearprog xteddy

# Echo the value of teddybearprog
Echo $[infostore.teddybearprog]
```

Removing an entry from the InfoStore is done with the **InfoStoreRemove** command.

InfoStoreRemove *key*

Removes an entry at the given *key* from the InfoStore. Example:

```
InfoStoreRemove teddybearprog
```

Nop

Does nothing. This is used to insert a blank line or separator in a menu. If the menu item specification is

```
AddToMenu MyMenu " " Nop
```

then a blank line is inserted. If it looks like

```
+ "" Nop
```

then a separator line is inserted. Can also be used as the double−click action for **Menu** or **Popup**.

PipeRead *command* [quiet]

Causes fvwm to read commands from the output of the *command*. This*command* is e xecuted by */bin/sh* as if you typed it on the command line. If the command consists of more than one word it must be quoted. Useful for building up dynamic menu entries based on a directories contents, for example. If the keyword *Quiet* follows the command no message is produced if the *command* is not found.

Example:

```
AddToMenu HomeDirMenu
PipeRead 'for i in $HOME/*; \
  do echo "+ $i Exec xterm −e vi $i"; done'
```

Note: The **PipeRead** changes the pointer to a watch cursor by default during execution. However, some commands, for example xwd, need to take control of the pointer themselves and do not work. To disable the watch cursor, use the command prior to **PipeRead**

```
BusyCursor Read off
```

The **PipeRead** command executes synchronously. If you want to **Exec** something, but need the command to run synchronously, you might do something like:

```
PipeRead 'command 1>&2'
```

The redirection causes any output from the program to go to stderr instead of being read as a

sequence of commands by fvwm. **PipeRead** returns 1 if the given command could be executed or −1 if not (see the section **Conditional Commands** for the meaning of return codes).

Read *filename* [quiet]
Causes fvwm to read commands from the file named *filename*. If the keyword *Quiet* follows the command no message is produced if the file is not found. If the file name does not begin with a slash ('/'), fvwm looks in the user's data directory, then the system data directory. The user's data directory is by default *$HOME/.fvwm*. It can be overridden by exporting *FVWM_USERDIR* set to any other directory. The**Read** command returns 1 if the gi ven file could be read or −1 if not (see the section **Conditional Commands** for the meaning of return codes).

SetEnv *variable value*
Set an environment variable to a new value, similar to the shell's export or setenv command. The *variable* and its *value* are inherited by processes started directly by fvwm. This can be especially useful in conjunction with the **FvwmM4** module. For example:

SetEnv height HEIGHT

makes the **FvwmM4** set variable *HEIGHT* usable by processes started by fvwm as the environment variable *$height*. If*value* includes whitespace, you should enclose it in quotes. If no *value* is given, the variable is deleted.

Silent *command*
A number of commands require a window to operate on. If no window was selected when such a function is invoked the user is asked to select a window. Sometimes this behavior is unwanted, for example if the function was called by a module and the window that was selected at first does not exist anymore. You can prevent this by putting **Silent** in front of the fvwm *command*. If a function that needs a window is called with **Silent** without a window selected, it simply returns without doing anything. If**Silent** is used on a user defined function it af fects all function and sub function calls until the original function exits.

Another usage of **Silent** is with binding commands **Key**, **PointerKey** and **Mouse**, this disables error messages.

Silent also disables the error message for non−existent commands. Note: This command is treated as a prefix to its *command*. Expansion of the command line is done as if**Silent** w as not there.

Examples:

Silent **Move** 0 0
Silent User_defined_function
do not complain on keyboards without "Help" key
Silent **Key** Help R A **Popup** HelpMenu

UnsetEnv [*variable*]
Unset an environment variable, similar to shell's export or unsetenv command. The *variable* then is removed from the environment array inherited by processes started directly by fvwm.

Wait *window*
This command is intended to be used in fvwm functions only. It causes execution of a function to pause until a new window matching *window* appears. This can be a window's name, class, or resource string. It may contain the wildcards '*' and '?', which are matched in the usual Unix filename manner. This is particularly useful in the "InitFunction" if you are trying to start windows on specific desktops:

AddToFunc InitFunction
+ I **Exec** exec xterm −geometry 80x64+0+0
+ I Wait xterm
+ I **GotoDesk** 0 2
+ I **Exec** exec xmh −font fixed −geometry \

```
    507x750+0+0
+ I Wait xmh
+ I GotoDesk 0 0
```

The above function starts an xterm on the current desk, waits for it to map itself, then switches to desk 2 and starts an xmh. After the xmh window appears control moves to desk 0.

Fvwm remains partially functional during a wait, but any input from the modules is queued up and processed only after the window appears or the command is aborted. For example, windows can not be focused with **FvwmIconMan** or **FvwmPager** during a wait.

You can escape from a **Wait** pause by pressing Ctrl-Alt-Escape (where Alt is the first modifier). To redefine this key sequence see the **EscapeFunc** command.

Conditional Commands

Conditional commands are commands that are only executed if certain conditions are met. Most conditional commands work on windows, like **Next**, **ThisWindow** or **All**. There is one conditional command, **Test**, that works on global conditions unrelated to windows. The syntax of the conditions is described below. For readability, the list of conditions is located at the end of this section.

Return Codes

All commands in this section (unless specifically stated for the command) also have a return code that can be 1 (if the condition was met) or 0 (if the condition was not met). Some commands may return −1 which means that an error occurred and the return code is useless. The **Break** command returns −2. Additionally, the return codes of commands run in a complex functions are passed to the invoking complex function. The return code is used by the **TestRc** command. Please refer to the commands' description for examples. The return code can also be accessed through the variable *$[cond.rc]*. Non conditional commands do not modify the return code of the last conditional command. Important note: return codes are only defined inside functions created with the **AddToFunc** command and are not inherited by sub functions. To run a command without altering the return code, the **KeepRc** command can be used.

The Ring of Windows

Fvwm stores windows in a ring internally. Think of the focused window as a cursor on the current position in the ring. The **Next** command and many other commands search forwards through the ring for a matching window, and **Prev** searches backwards. The windows in the ring are either ordered by creation time (if the *!FPSortWindowlistByFocus*, *NeverFocus* or *MouseFocus* styles are used) or by the last time they had the focus.

List of Conditional Commands

All [*options*] [(*conditions*)] *command*

Execute *command* on all windows meeting the conditions. It returns 1 if any window matches the condition and 0 otherwise. The execution starts at the top of the window ring and continues towards the bottom. The *options* can be any combination of *Reverse* and *UseStack*. If the option*Re verse* is given the execution order is reversed. The option *UseStack* makes All use the stacking order instead of the window ring when walking through windows. See the**Conditions** section for a list of conditions.

This command implies the conditions *CirculateHit*, *CirculateHitIcon* and *CirculateHitShaded*. They can be turned off by specifying *!CirculateHit* etc. explicitly.

Any [(*conditions*)] *command*

Performs *command* if any window which satisfies all *conditions* exists. The command is run in the context of the root window. See the**Conditions** section for a list of conditions.

Break [*levels*]

If the break command is used in a function, function execution is terminated immediately. Further commands of the function are not processed. Normally, all nested invocations of complex functions are left. An optional integer number *levels* may be given to break out of the given number of nested functions and continue execution of a higher level function.

The **Break** command always has the return code −2. Example:

```
AddToFunc PickWindowRaiseAndDeiconify
+ I Pick
+ I TestRc (Error) Break
+ I Raise
+ I Iconify off
```

Current [(*conditions*)] *command*
Performs *command* on the currently focused window if it satisfies all *conditions*. See the **Conditions** section for a list of conditions.

This command implies the conditions *CirculateHit*, *CirculateHitIcon* and *CirculateHitShaded*. They can be turned off by specifying *!CirculateHit* etc. explicitly.

Direction [FromPointer] *direction* [(*conditions*)] *command*
Performs *command* (typically **Focus**) on a window in the given direction which satisfies all *conditions*. Normally, the center of the currently focused window or the context window in which the command was invoked is taken as the starting point. Lacking such a window, or when the *FromPointer* option is given, the current position of the pointer is taken as the starting point. The *direction* may be one of "North", "Northeast", "East", "Southeast", "South", "Southwest", "West", "Northwest" and "Center". Which window **Direction** selects depends on angle and distance between the center points of the windows. Closer windows are considered a better match than those farther away. The *Center* direction simply selects the window closest to the starting point. Returns −1 if an invalid direction was given. See the**Conditions** section for a list of conditions.

KeepRc *command*
Runs the *command* but does not alter the return code of the previous command. Note: **KeepRc** is treated as a prefix to its *command*. Expansion of the command line is done as if **KeepRc** was not there.

Next [(*conditions*)] *command*
Performs *command* (typically **Focus**) on the next window which satisfies all *conditions*. If the command is running in a window context, it starts looking for a matching window from there. Otherwise it starts at the focused window. See**Conditions** section for a list of conditions.

None [(*conditions*)] *command*
Performs *command* if no window which satisfies all *conditions* exists. The command is run in the context of the root window. Returns 1 if no window matches the conditions and 0 otherwise. See **Conditions** section for a list of conditions.

This command implies the conditions *CirculateHit*, *CirculateHitIcon* and *CirculateHitShaded*. They can be turned off by specifying *!CirculateHit* etc. explicitly.

NoWindow *command*
Performs *command*, but removes the window context if any. This is not really a conditional command, but a prefix that may be useful in menu items that should operate without a window even if such menu is bound to window decorations.

Pick [(*conditions*)] *command*
Pick works like **Function** if invoked in the context of a window. If invoked in the root window, it first asks the user to pick a window and then executes the *command* in the context of that window. This avoids annoying multiple selections with complex functions. The command is executed only if the given *conditions* are met. Returns −1 if no window was selected. See **Conditions** section for a list of conditions.

This command implies the conditions *CirculateHit*, *CirculateHitIcon* and *CirculateHitShaded*. They can be turned off by specifying *!CirculateHit* etc. explicitly.

PointerWindow [(*conditions*)] *command*
Performs *command* if the window under the pointer satisfies all *conditions*. Returns −1 if there is no window under the pointer. See**Conditions** section for a list of conditions.

This command implies the conditions *CirculateHit*, *CirculateHitIcon* and *CirculateHitShaded*. They can be turned off by specifying *!CirculateHit* etc. explicitly.

Prev [(*conditions*)] *command*
Performs *command* (typically **Focus**) on the previous window which satisfies all *conditions*. If the command is running in a window context, it starts looking for a matching window from there. Otherwise it starts at the focused window. See**Conditions** section for a list of conditions.

ScanForWindow [FromPointer] *dir1 dir2* [(*conditions*)] *command*
Performs *command* (typically **Focus**) on a window in the given direction which satisfies all *conditions*. Normally, the center of the currently focused window or the context window in which the command was invoked is taken as the starting point. Lacking such a window, or when the *FromPointer* option is given, the current position of the pointer is taken as the starting point. The direction *dir1* may be one of "North", "NorthEast", "East", "SouthEast", "South", "SouthWest", "West", and "NorthWest". Which window **ScanForWindow** selects depends first on the position along the primary axis given by *dir1*. If any windows have the exact same coordinate along the primary axis, the secondary direction is used to order the windows. The direction*dir2* may be one of the same set of values as *dir1*. If*dir2* is not perfectly perpendicular to *dir1*, ScanF orWindow returns a failure. When using ScanForWindow repeatedly with the same arguments, it is guaranteed that all windows matching the conditions will eventually be found. If the focus reaches a limit along the primary axis, it will wrap around to the opposite side. Returns −1 if an invalid direction was given. See**Conditions** section for a list of conditions.

Test [(*test−conditions*)] *command*
Performs *command* if all *test−conditions* are satisfied. The *test−conditions* are keywords with possible arguments from the list below and are separated by commas or whitespace. They include: *Version operator x.y.z*, *EnvIsSet varname*, *EnvMatch varname pattern*, *EdgeHasPointer direction*, *EdgeIsActive direction*, *Start*, *Init*, *Restart*, *Exit*, *Quit*, *ToRestart*, *True*, *False*, *F*, *R*, *W*, *X* and *I*. A test−condition prefixed with "!" is negated.

The *Version operator x.y.z* test−condition is fulfilled if the logical condition of the expression is true. Valid *operator* values are: >=, >, <=, <, == and *!=*.

Example:

Test (Version >= 2.5.11) **Echo** 2.5.11 or later.

The *EnvIsSet varname* test−condition is true if the given environment variable is set. The *EnvMatch varname pattern* test−condition is true if *pattern* matches the given environment or infostore variable value. (See**Inf oStoreAdd**). The pattern may contain special "*" and "?" chars. The "varname" is coded without the leading dollar sign ($).

The *EdgeHasPointer* [*direction*] test−condition is true if the edge in the given direction currently contains the pointer. The*Edg eIsActive* [*direction*] test−condition is true if the edge in the given direction currently is active. An edge is active, and can contain a pointer if either a command is bound to it or edge scroll is available in that direction. The direction may be one of
Any, *North*, *Top*, *Up*, *West*, *Left*, *South*, *Bottom*,
Down, *Right* and *East*. If no direction is specified*Any* is assumed.

The *Start* test−condition is the same as either *Init* or *Restart*. It is only true on startup or restart prior and during **StartFunction** execution. The*Exit* test−condition is the same as either *Quit* or *ToRestart*. It is only valid on shutdown during **ExitFunction** function

execution.

The *True* and *False* test−conditions are unconditionally true and false.

Additionally, if a test−condition name is not recognized, the Error return code is set and the command is not executed.

The *F file*, *R file*, *W file*, *X file* and *I file* test−conditions test for existence of the given [F]ile (possibly with [R]ead/[W]rite permissions), e[X]ecutable (in *$PATH*), or the [I]mage (in ImagePath).

Example:

```
AddToFunc StartFunction I Test (Init) Exec exec xterm

AddToFunc VerifyVersion
+ I Test (Version 2.5.*) Echo 2.5.x detected
+ I TestRc (NoMatch) \
        Test (!Version 2.6.*) Echo Future version
+ I TestRc (NoMatch) \
        Echo 2.6.x is detected

Test (F $[FVWM_USERDIR]/local−config) Read local−config
Test (X xterm−utf16) Exec exec xterm−utf16
```

TestRc [([!]*returncode*)] *command*

Performs *command* if the last conditional command returned the value *returncode*. Instead of the numeric values 0 (no match), 1 (match), −1 (error), and −2 (break) the symbolic names "*NoMatch*", "*Match*", "*Error*" and "*Break*" can be used. If no *returncode* is given, the default 0 is assumed. If the return code is prefixed with '!', the command is executed if *returncode* does not match the value returned by the conditional command. The **TestRc** command can only be used inside functions. If the *command* is another conditional command, the previous return code is replaced by the new one. Example:

```
AddToFunc ToggleXterm
+ I All (my_xtermwindow) Close
+ I TestRc (NoMatch) Exec xterm −T my_xtermwindow
```

ThisWindow [(*conditions*)] command

ThisWindow executes the specified *command* in the context of the current operand window. If there is no operand window (it is invoked in the root window), the command is ignored. **ThisWindow** is never interactive. The command is executed only if the given *conditions* are met. It returns −1 if used outside a window context. See**Conditions** section for a list of conditions.

This command implies the conditions *CirculateHit*, *CirculateHitIcon* and *CirculateHitShaded*. They can be turned off by specifying "!CirculateHit" etc. explicitly.

WindowId [*id*] [(*conditions*)] | [root [*screen*]] *command*

The **WindowId** command looks for a specific window *id* and runs the specified *command* on it. The second form of syntax retrieves the window id of the root window of the given *screen*. If no*scr een* is given, the current screen is assumed. The window indicated by *id* may belong to a window not managed by fvwm or even a window on a different screen. Although most commands can not operate on such windows, there are some exceptions, for example the **WarpToWindow** command. Returns −1 if no window with the given id exists. See**Conditions** section for a list of conditions.

This command implies the conditions *CirculateHit*, *CirculateHitIcon* and *CirculateHitShaded*. They can be turned off by specifying *!CirculateHit* etc. explicitly.

Examples:

WindowId 0x34567890 **Raise**
WindowId root 1 **WarpToWindow** 50 50
WindowId $0 (Silly_Popup) **Delete**

In the past this command was mostly useful for functions used with the **WindowList** command, or for selective processing of **FvwmEvent** calls (as in the last example), but currently these handler functions are called within a window context, so this command is not really needed in these cases. Still it may be useful if, for example, the window id should be stored in the environment variable for a further proceeding.

Pick SetEnv BOOKMARKED_WINDOW $[w.id]
WindowId $[BOOKMARKED_WINDOW] **WarpToWindow**

Conditions

The *conditions* that may be given as an argument to any conditional command are a list of keywords separated by commas, enclosed in parentheses. Unless stated otherwise, conditional commands accept all the conditions listed below. Note that earlier versions of fvwm required the conditions to be separated by whitespace instead of commas and enclosed in brackets instead of parentheses (this is still supported for backward compatibility).

In addition, the *conditions* may include one or more window names to match to. If more than one window name is given, all of them must match. The window name, icon name, class, and resource are considered when attempting to find a match. Each name may include the wildcards '*' and '?', and may consist of two or more alternatives, separated by the character '|', which acts as an OR operator. (If OR operators are used, they must not be separated by spaces from the names.) Each window name can begin with '!', which prevents *command* if any of the window name, icon name, class or resource match. However, '!' must not be applied to individual names in a group separated by OR operators; it may only be applied to the beginning of the group, and then it operates on the whole group.

Examples:

Next ("Netscape|konqueror|Mozilla*") **WarpToWindow** 99 90

This goes to the next web browser window, no matter which of the three named web browsers is being used.

Next ("Mozilla*", "Bookmark*") **WarpToWindow** 99 90

This goes to Mozilla's bookmark manager window, ignoring other Mozilla windows and other browsers' bookmark windows.

All ("XTerm|rxvt", !console) **Iconify**

This iconifies all the xterm and rxvt windows on the current page, except that the one named "console" (with the −name option to xterm) is excluded.

Next (!"**FvwmPager**|**FvwmForm***|**FvwmButtons**") **Raise**
Next (!**FvwmPager**, !**FvwmForm***, !**FvwmButtons**) **Raise**

These two commands are equivalent; either one raises the next window which is not one of the named fvwm modules.

Any condition can be negated by using a an exclamation mark ('!') directly in front of its name.

AcceptsFocus, *AnyScreen*, *CirculateHit*, *CirculateHitIcon*, *CirculateHitShaded*, *Closable*, *CurrentDesk*, *CurrentGlobalPage*, *CurrentGlobalPageAnyDesk*, *CurrentPage*, *CurrentPageAnyDesk*, *CurrentScreen*, *Desk*, *FixedPosition*, *FixedSize*, *Focused*, *HasHandles*,

HasPointer, Iconic, Iconifiable, Layer [n], Maximizable, Maximized, Overlapped, PlacedByButton n, PlacedByButton3, PlacedByFvwm, Raised, Shaded, State n, Sticky, StickyAcrossDesks, StickyAcrossPages, StickyIcon, StickyAcrossDesksIcon, StickyAcrossPagesIcon, Transient, Visible.

The *AcceptsFocus* condition excludes all windows that do not want the input focus (the application has set the "Input hints" for the window to False) and do not use the *Lenience* option of the **Style** command. Also, all windows using the *NeverFocus* style are ignored. Note: *!Lenience* is equivalent to the deprecated option *NoLenience*.

With the *AnyScreen* condition used together with any of the *Current...* conditions, windows that do not intersect the Xinerama screen containing the mouse pointer are considered for a match too. For example:

```
# Focus next window on current page,
# regardless of Xinerama screen
Next (CurrentPage, AnyScreen) Focus
```

The *CirculateHit* and *CirculateHitIcon* options override the *CirculateSkip* and *CirculateSkipIcon* **Style** attributes for normal or iconic windows. The*Cir culateHitShaded* option overrides the *CirculateSkipShaded* **Style.** All three options are turned on by default for the **Current** command. They can be turned off by specifying *!CirculateHit* etc. explicitly. Note: Do not confuse these conditions with the style options of the same name. Specifically,

```
Style foo CirculateSkip
Next (foo, CirculateHit) ...
```

is not the same as

```
Style foo CirculateHit ...
Next (foo)
```

The prior selects windows with the name foo only in the Next command. In the second example, these windows are always matched in all conditional commands.

The *Closable* condition matches only windows that are allowed to be closed.

The *CurrentDesk* condition matches only windows that are on the current desk.

The *CurrentGlobalPage* condition matches only windows that are on the current page of the current desk, regardless of whether Xinerama support is enabled or not. This condition implicitly activates the *CurrentDesk* condition.

The *CurrentGlobalPageAnyDesk* condition matches only windows that are on the current page of any desk, regardless of whether Xinerama support is enabled or not.

The *CurrentPage* condition matches only windows that are on the current page of the current desk. If Xinerama support is enabled, it only matches windows that are at least partially on the Xinerama screen containing the mouse pointer. This condition implicitly activates the *CurrentDesk* condition.

The *CurrentPageAnyDesk* and *CurrentScreen* conditions matches only windows that are on the current page of any desk. If Xinerama support is enabled, they only match windows that are at least partially on the Xinerama screen containing the mouse pointer.

The *Screen [n]* condition matches only windows which are on the specified Xinerama screen.

The *Desk [n]* condition matches only windows which are on the specified desk.

The *FixedPosition* condition excludes all windows that do not have a fixed position, either set through WM hints or the **Style** option *FixedPosition*. Example:

```
DestroyFunc ToggleFixedGeometry
AddToFunc   ToggleFixedGeometry
```

+ I **Pick** (FixedPosition) \
    WindowStyle VariablePosition, VariableSize
+ I **TestRc** (NoMatch) **WindowStyle** FixedPosition, FixedSize

The *FixedSize* condition excludes all windows that do not have a fixed size, either set through WM hints or the **Style** option *FixedSize*.

The *Focused* matches on the window that currently has the keyboard focus. This is not useful for the **Current** command but can be used with the other conditional commands.

The *HasHandles* condition excludes all windows that do not have resize handles.

The *HasPointer* condition excludes all windows that do not contain the pointer.

The *Iconic* condition matches only iconic windows.

The *Iconifiable* condition matches only windows that are allowed to be iconified.

The *Layer [n]* condition matches only windows on the specified layer. The optional argument of the *Layer* condition defaults to the layer of the focused window. The negation *!Layer* switches off the *Layer* condition.

The *Maximizable* condition matches only windows that are allowed to be maximized.

The *Maximized* condition matches only maximized windows.

The *Fullscreen* condition matches only fullscreen windows.

The *Overlapped* condition matches only windows that are overlapped by other windows on the same layer (or unmanaged windows if the option *RaiseOverUnmanaged* of the **BugOpts** command is used). Note that this condition can be slow if you have many windows or if RaiseOverUnmanaged is used and the connection to the X server is slow.

The *PlacedByButton n* condition is fulfilled if the last interactive motion of the window (with the **Move** command or as *ManualPlacement*) was ended by pressing mouse button *n*. Example:

Mouse 1 T A **Function** MoveWindow

DestroyFunc MoveWindow
AddToFunc MoveWindow
+ C **Move**
+ C **ThisWindow** (PlacedByButton 5) **WindowShade** off
+ C **TestRc** (Match) **Maximize** on 0 100
+ C **ThisWindow** (PlacedByButton 4) **WindowShade** on

The *PlacedByButton3* condition has the same meaning as *PlacedByButton* 3. It remains only for backward compatibility.

The *PlacedByFvwm* condition excludes all windows that have been placed manually or by using the user or program position hint.

The *Raised* conditions matches only windows that are fully visible on the current viewport and not overlapped by any other window.

The *Shaded* conditions matches only shaded windows (see **WindowShade** command).

The *State n* or *!State n* conditions match only windows with the specified integer state set (or unset). See the**State** command for details. The ar gument may range from 0 to 31.

The *Sticky*, *StickyAcrossDesks* and *StickyAcrossPages* match only windows that are currently sticky, sticky across all desks or sticky across all pages. Please refer to the **Style** options with the same name and the commands **Stick**, **StickAcrossDesks** and **StickAcrossPages** for details.

The *StickyIcon*, *StickyAcrossDesksIcon* and *StickyAcrossPagesIcon* match only windows that become sticky, sticky across all desks or sticky across all pages when they are in iconified state.

The *Transient* condition matches only windows that have the "transient" property set by the application. This it usually the case for application popup menus and dialogs. The**FvwmIdent** module can be used to find out whether a specific window is transient.

The *Visible* condition matches only windows that are at least partially visible on the current viewport and not completely overlapped by other windows.

Module Commands

Fvwm maintains a database of module configuration lines in a form

*****<*ModuleName*>**:** <*Config−Resource*>

where <*ModuleName*> is either a real module name or an alias.

This database is initially filled from config file (or from output of **−cmd** config command), and can be later modified either by user (via **FvwmCommand**) or by modules.

When modules are run, they read appropriate portion of database. (The concept of this database is similar to one used in X resource database).

Commands for manipulating module configuration database are described below.

***** *module_config_line*

Defines a module configuration. *module_config_line* consists of a module name (or a module alias) and a module resource line. The new syntax allows a delimiter, a colon and optional spaces, between the module name and the rest of the line, this is recommended to avoid conflicts.

```
*FvwmPager: WindowBorderWidth 1
*FvwmButtons−TopRight: Geometry 100x100−0+0
*FvwmButtons−Bottom: Geometry +0−0
```

DestroyModuleConfig *module_config*

Deletes module configuration entries, so that new configuration lines may be entered instead. This also sometimes the only way to turn back some module settings, previously defined. This changes the way a module runs during a fvwm session without restarting. Wildcards can be used for portions of the name as well.

The new non−conflicting syntax allows a delimiter, a colon and optional spaces between the module name and the rest of the line. In this case a module name (or alias) can't have wildcards.

```
DestroyModuleConfig FvwmButtons*
DestroyModuleConfig FvwmForm: Fore
DestroyModuleConfig FvwmIconMan: Tips*
```

KillModule *modulename* [*modulealias*]

Causes the module which was invoked with name *modulename* to be killed. The name may include wildcards. If *modulealias* is given, only modules started with the given alias are killed.

```
# kill all pagers
KillModule FvwmPager

Module FvwmEvent SoundEvent
KillModule FvwmEvent SoundEvent
```

Module *modulename* [*moduleparams*]

Specifies a module with its optional parameters which should be spawned. Currently several modules, including **FvwmButtons**, **FvwmEvent**, **FvwmForm**, **FvwmPager**, **FvwmScript** support aliases. Aliases are useful if more than one instance of the module should be spawned. Aliases may be configured separately using ***** syntax. To start a module **FvwmForm** using an alias *MyForm*, the following syntax may be used:

```
Module FvwmForm MyForm
```

At the current time the available modules (included with fvwm) are **FvwmAnimate** (produces animation effects when a window is iconified or de−iconified), **FvwmAuto** (an auto raise module), **FvwmBacker** (to change the background when you change desktops), **FvwmBanner** (to display a spiffy XBM, XPM, PNG or SVG), **FvwmButtons** (brings up a customizable tool bar), **FvwmCommandS** (a command server to use with shell's FvwmCommand client), **FvwmConsole** (to execute fvwm commands directly), **FvwmCpp** (to preprocess your *config* with cpp), **FvwmEvent** (trigger various actions by events), **FvwmForm** (to bring up dialogs), **FvwmIconMan** (a flexible icon manager), **FvwmIdent** (to get window info), **FvwmM4** (to preprocess your *config* with m4), **FvwmPager** (a mini version of the desktop), **FvwmPerl** (a Perl manipulator and preprocessor), **FvwmProxy** (to locate and control obscured windows by using small proxy windows), **FvwmRearrange** (to rearrange windows), **FvwmScript** (another powerful dialog toolkit), These modules have their own man pages. There may be other modules out on there as well.

Modules can be short lived transient programs or, like **FvwmButtons** , can remain for the duration of the X session. Modules are terminated by the window manager prior to restarts and quits, if possible. See the introductory section on modules. The keyword **Module** may be omitted if *modulename* is distinct from all fvwm commands.

ModuleListenOnly *modulename* [*moduleparams*]

This command works like the **Module** command, but fvwm never sends any messages to the module. This may be handy to write a module as a shell script that is triggered by external events without the burden to answer packets sent by fvwm. For example, a module written as a shell script may change labels of the **FvwmButtons** module to implement a simple clock.

ModulePath *path*

Specifies a colon separated list of directories in which to search for modules. To find a module, fvwm searches each directory in turn and uses the first file found. Directory names on the list do not need trailing slashes.

The **ModulePath** may contain environment variables such as *$HOME* (or *${HOME}*). Further, a '+' in the *path* is expanded to the previous value of the *path*, allowing easy appending or prepending to the *path*.

For example:

```
ModulePath ${HOME}/lib/fvwm/modules:+
```

The directory containing the standard modules is available via the environment variable *$FVWM_MODULEDIR*.

ModuleSynchronous [Expect *string*] [Timeout *secs*] *modulename*

The **ModuleSynchronous** command is very similar to **Module**. Fvwm stops processing any commands and user input until the module sends a string beginning with "NOP FINISHED STARTUP" back to fvwm. If the optional *Timeout* is given fvwm gives up if the module sent no input back to fvwm for *secs* seconds. If the *Expect* option is given, fvwm waits for the given *string* instead. **ModuleSynchronous** should only be used during fvwm startup to enforce the order in which modules are started. This command is intended for use with the (currently hypothetical) module that should be in place before other modules are started.

Warning: It is quite easy to hang fvwm with this command, even if a timeout is given. Be extra careful choosing the string to wait for. Although all modules in the fvwm distribution send back the "NOP FINISHED STARTUP" string once they have properly started up, this may not be the case for third party modules. Moreover, you can try to escape from a locked **ModuleSynchronous** command by using the key sequence Ctrl-Alt-Escape (see the **EscapeFunc**).

ModuleTimeout *timeout*
Specifies how many seconds fvwm waits for a module to respond. If the module does not respond within the time limit then fvwm kills it. *timeout* must be greater than zero, or it is reset to the default value of 30 seconds.

SendToModule *modulename string*
Sends an arbitrary string (no quotes required) to all modules, whose alias or name matching *modulename*, which may contain wildcards. This only makes sense if the module is set up to understand and deal with these strings though. Can be used for module to module communication, or implementation of more complex commands in modules.

Session Management Commands

Quit
Exits fvwm, generally causing X to exit too.

QuitScreen
Causes fvwm to stop managing the screen on which the command was issued.

Restart [*window_manager* [*params*]]
Causes fvwm to restart itself if *window_manager* is left blank, or to switch to an alternate window manager (or other fvwm version) if *window_manager* is specified. If the window manager is not in your default search path, then you should use the full path name for *window_manager*.

This command should not have a trailing ampersand. The command can have optional parameters with simple shell−like syntax. You can use ˜ (is expanded to the user's home directory) and environmental variables *$VAR* or *${VAR}*. Here are several examples:

```
Key F1 R N Restart
Key F1 R N Restart fvwm −s
Key F1 R N Restart ˜/bin/fvwm −f $HOME/.fvwm/main
Key F1 R N Restart fvwm1 −s −f .fvwmrc
Key F1 R N Restart xterm −n '"X console"' \
  −T \"X\ console\" −e fvwm1 −s
```

If you need a native restart, we suggest only to use **Restart** command without parameters unless there is a reason not to. If you still use an old command 'Restart fvwm2' that was correct in 2.2.x, all current command line arguments are lost. On a restart without parameters or with −−pass−args, they are preserved. Here are some cases when 'Restart fvwm2' or 'Restart fvwm' cause troubles:

```
* running fvwm under a session manager
* running fvwm with multi headed displays
* having command line arguments, like
  −f themes−rc or −cmd
* if the first fvwm2 in the $PATH is a
  different one
```

This is why we are issuing a warning on an old usage. If you really want to restart to fvwm with no additional arguments, you may get rid of this warning by using "Restart fvwm −s" or "Restart /full/path/fvwm".

Note, currently with multi headed displays, restart of fvwms on different screens works independently.

Restart −−pass−args *window_manager*
The same as **Restart** without parameters but the name for the current window manager is replaced with the specified *window_manager* and original arguments are preserved.

This command is useful if you use initial arguments like

−cmd FvwmCpp

and want to switch to another fvwm version without losing the initial arguments.

Restart −−dont−preserve−state [*other−params*]
The same as

Restart [*other−params*]

but it does not save any window states over the restart.

Without this option, **Restart** preserves most per−window state by writing it to a file named *.fs−restart−$HOSTDISPLAY* in the user's home directory.

SaveSession
Causes a session manager (if any) to save the session. This command does not work for xsm, it seems that xsm does not implement this functionality. Use Unix signals to manage xsm remotely.

SaveQuitSession
Causes a session manager (if any) to save and then shutdown the session. This command does not work for xsm, it seems that xsm does not implement this functionality. Use Unix signals to manage xsm remotely.

Colorsets

Colorsets are a powerful method to control colors. Colorsets create appearance resources that are shared by fvwm and its modules. When a colorset is modified all parts of fvwm react to that change. A colorset includes a foreground color, background color, shadow and highlight color (often based on the background color), background face (this includes images and all kinds of gradients). There is a way to render background face and specify other color operations.

In the 2.4.x versions a special module **FvwmTheme** was introduced to manage colorsets. Starting with the 2.5.x beta version, the **FvwmTheme** functionality was moved to the core fvwm, so this module became obsolete. In 2.6.7 the**FvwmTheme** module w as removed.

The old syntax:

DestroyModuleConfig FvwmTheme: *
*__FvwmTheme__: __Colorset__ 0 fg black, bg rgb:b4/aa/94
*__FvwmTheme__: __Colorset__ 1 fg black, bg rgb:a1/b2/c8

corresponds to the new syntax:

CleanupColorsets
Colorset 0 fg black, bg rgb:b4/aa/94
Colorset 1 fg black, bg rgb:a1/b2/c8

Colorset *num* [*options*]
Creates or modifies colorset *num*. Colorsets are identified by this number. The number can start at zero and can be a very large number.

Warning: The highest colorset number used determines memory consumption. Thus, if you define 'Colorset 100000', the memory for 100001 colorsets is used. Keep your colorset numbers as small as possible.

By convention, colorsets are numbered like this:

0 = Default colors
1 = Inactive windows
2 = Active windows
3 = Inactive menu entry and menu background
4 = Active menu entry

```
# 5 = greyed out menu entry (only bg used)
# 6 = module foreground and background
# 7 = hilight colors
```

If you need to have more colors and do not want to reinvent the wheel, you may use the convention used in fvwm−themes, it defines the meaning of the first 40 colorsets for nearly all purposes:

http://fvwm−themes.sourceforge.net/doc/colorsets

Each colorset has four colors, an optional pixmap and an optional shape mask. The four colors are used by modules as the foreground, background, highlight and shadow colors. When a colorset is created it defaults to a foreground of black and background of gray. The background and foreground are marked as "average" and "contrast" (see later) so that just specifying a pixmap or gradient gives sensible results.

options is a comma separated list containing some of the keywords: fg, Fore, Foreground, bg, Back, Background, hi, Hilite, Hilight, sh, Shade, Shadow, fgsh, Pixmap, TiledPixmap, AspectPixmap, Transparent, RootTransparent, Shape, TiledShape, AspectShape, NoShape, ?Gradient, Tint, fgTint, bgTint, Alpha, fgAlpha, Dither, NoDither, IconTint, IconAlpha, Plain.

fg, *Fore* and *Foreground* take a color name as an argument and set the foreground color. The special name *Contrast* may be used to select a color that contrasts well with the background color. To reset the foreground color to the default value you can simply omit the color name.

bg, *Back* and *Background* take a color name as an argument and set the background color. It also sets the highlight and shadow colors to values that give a 3d effect unless these have been explicitly set with the options below. The special name*A verage* may be used to select a color that is the average color of the pixmap. If the pixmap is tinted with the *Tint* option, the tint is not taken in account in the computation of the average color. You should use the *bgTint* option to get the "real" average color. The background color is reset to the default value if the color name is omitted.

hi, *Hilite* and *Hilight* take a color name as an argument and set the highlight color. If the highlight color is not explicitly set, the default is to calculate it from the background color. To switch back to the default behavior the color name can be omitted.

sh, *Shade* and *Shadow* take a color name as an argument and set the shadow color. If the shadow color is not explicitly set, the default is to calculate it from the background color. To switch back to the default behavior the color name can be omitted.

fgsh takes a color name as an argument and sets the color used by the shadowing font effect. See the **Font Shadow Effects** section of the fvwm man page. By default this color is computed from the foreground and background colors. To switch back to the default the color name can be omitted.

Pixmap, *TiledPixmap* and *AspectPixmap* take a file name as an argument, search the **ImagePath** and use it as the background pixmap. Any transparent parts are filled with the background color. Not specifying a file name removes any existing image from the colorset. *TiledPixmap* produces repeated copies of the image with no scaling, *Pixmap* causes the image to be stretched to fit whatever object the colorset is applied to and *AspectPixmap* stretches to fit but retains the image aspect ratio.

Transparent creates a transparent background pixmap. The pixmap is used as a window background to achieve root transparency. For this you should use the *ParentalRelativity* option to the **Style** command. A subsequent root background change may be detected or not, this depends on the program used to set the background. If you use **fvwm−root**, **xsetbg** (xli), **FvwmBacker** with solid or colorset colors or a recent version of **Esetroot** (>= 9.2) a background change is detected. If background changes are not detected (e.g., if you use**xv** or **xsetr oot**) you can force detection by using the **−d** option of fvwm−root:

xv −root −quit mybg.png; fvwm−root −d

Due to the way X implements transparency no guarantees can be made that the desired effect can be achieved. The application may even crash. If you experience any problems with this option, do not use it.

Using outline move and resize (see the **OpaqueMoveSize** command and the *ResizeOpaque* **Style** option) as well as setting the *WindowShadeShrinks* style may help. The transparency achieved with *Transparent* depends on whether the colorset is applied to the foreground or the background of a window. In the second case the transparency is relative to the parent window of the window on which the colorset is defined. For example:

Colorset 12 VGradient 200 grey30 grey60
Colorset 17 Transparent
*__FvwmIconMan__: Colorset 12
*__FvwmIconMan__: PlainColorset 17

gives an IconMan with a vertical grey gradient background and the buttons use the background (by transparency). To obtain a (root) transparent IconMan:

Colorset 12 Transparent
Colorset 17 Transparent
Colorset 18 Transparent
Colorset 19 Transparent

*__FvwmIconMan__: Colorset 12
*__FvwmIconMan__: PlainColorset 17
*__FvwmIconMan__: FocusColorset 18
*__FvwmIconMan__: IconColorset 19

The Colorset IconMan option defines the IconMan window background, but the PlainColorset and the FocusColorset are drawn on the foreground. So, the transparency of the IconMan buttons is achieved by drawing nothing. Now if this IconMan is swallowed in an FvwmButtons as:

__FvwmButtons__:(Colorset 10, Swallow "FvwmIconMan" 'FvwmIconMan')

then, **FvwmIconMan** becomes a child of **FvwmButtons** and it is transparent relative to **FvwmButtons**. So, in this case**FvwmIconMan** uses Colorset 10 as background. If you w ant root transparency use the *RootTransparent* option. **FvwmButtons FvwmIconMan**, and **FvwmIdent**, are relatively simple. There is one main colorset option which defines the background of the window and the other colorsets (if any) are drawn on the foreground. The case of **FvwmProxy** is simpler, the two colorsets refer to the window backgrounds. **FvwmPager** is more complicated as almost everything in the pager are windows with some parental relations (the mini windows are the child and the desktops are the parents and all this is complicated by the hilighted page). So, the colorsets apply to the background of these windows. You should experiment. For **FvwmForm** and **FvwmScript** the situation is similar. There is a main window (a child of the root window) which corresponds to the main colorset and most of the widgets are windows which are children of the main window. *Tint* may work or not with the *Transparent* option. When the colorset is drawn on the foreground *Tint* should work. In some cases, tinting may be very slow. Tinting may work with fvwm menu (without animation). Tinting may work better if your X server has backing store enabled (try xdpyinfo to see if this the case). There is a chance that the backing store support of your X server does not work well with the terrible hack used to Tint the ParentRelative Pixmap. So, to get tinted root transparency it is more safe to use the *RootTransparent* option.

RootTransparent [*buffer*] creates a root transparent background. To make this option work, you must use an **Esetroot** compatible program, fvwm−root with the −−retain−pixmap option or **FvwmBacker** with the RetainPixmap option (and colorset or solid backgrounds). The *buffer*

keyword is useful only when the *Tint* option is used too. This speeds up creation of windows which use the colorset (useful for fvwm menus) at the cost of memory usage. It also speeds up opaque move and resize which can be unacceptably slow without *buffer*. However, this option may add a lot of memory to your X server (depending on the size of the image used to set the background). In summary, using outline move and resize for modules which use such a colorset may be a good idea.

Shape, *TiledShape* and *AspectShape* take a file name as an argument, search the **ImagePath** and use it as the shape bitmap. *TiledShape* produces repeated copies of the bitmap with no scaling, *Shape* causes the bitmap to be stretched to fit whatever object the colorset is applied to and *AspectShape* stretches to fit but retains the bitmap aspect ratio. If the file is a pixmap in xpm format the shape mask (all opaque pixels) of the pixmap is used. For png and svg images, the shape mask is equivalent to all not completely transparent pixels (alpha > 0).

Warning
Due to the way X11 implements shapes you cannot take back making windows shaped. You may have to restart fvwm or the shaped application.

?Gradient ... creates a pixmap and stretches it to fit the window. *?Gradient* may be one of *HGradient, VGradient, DGradient, BGradient, SGradient, CGradient, RGradient* or *YGradient*. The gradient types are as follows: H is horizontal; V is vertical; D is diagonal from top left to bottom right; B is a backwards diagonal from bottom left to top right; S is concentric squares; C is concentric circles; R is a radar like pattern and Y is a Yin Yang style (but without the dots). Please refer to the **Color Gradients** section for the syntax of gradients.

Tint takes 2 arguments, a color and a percentage between 0 and 100. It causes the image defined using *?Pixmap* or *?Gradient* to be tinted with the specified color using the percentage. If the image is transparent *Tint* tints only the image part. Unfortunately, a colorset background specified using the *Transparent* option can give strange results. See the *Transparent* option for details. With no arguments this option removes the tint.

fgTint takes 2 arguments, a color and a percentage between 0 and 100. It causes the color defined using *fg* to be tinted with the specified color using the percentage. With no arguments this option removes the tint.

bgTint takes 2 arguments, a color and a percentage between 0 and 100. It causes the color defined using *bg* to be tinted with the specified color using the percentage. If the *sh* and *hi* colors are not specified, they are recomputed from the tinted bg color. With no arguments this option removes the tint.

Alpha takes a percentage between 0 and 100 as an argument. It causes fvwm to merge the image defined using *?Pixmap* or *?Gradient* with the *bg* color using the percentage. If the percentage is 0 the image is hidden and if it is 100 the image is displayed as usual (no merge). The default is 100 and it is restored if no argument is given.

fgAlpha takes a percentage between 0 and 100 as an argument. It causes fvwm to merge the text and the colorset background using the percentage. If the percentage is 0 the text is hidden and if it is 100 the text is displayed as usual (no merge). This option has an effect only with fonts loaded by Xft, see the **Font Names and Font Loading** section. The default is 100 and it is restored if no argument is given.

Dither causes fvwm to dither the image defined using *?Pixmap* or *?Gradient*. This is useful only with displays with depth less than or equal to 16 (i.e., on displays which can only display less than 65537 colors at once). The dithering effect lets you simulate having more colors available that you actually have. *NoDither* causes fvwm to do not dither the images. *Dither* is the default if the depth is less than or equal to 8 (a screen with 256 colors or less). In depth 15 (32768 colors) and 16 (65536 colors), the default is *NoDither*, however this effect can be useful with images which contain a lot of close colors. For example a fine gradient looks more smooth.

IconTint takes 2 arguments, a color and a percentage between 0 and 100. It causes fvwm or a

module to tint the "icons" which are rendered into the colorset background with the specified color using a percentage. Here "icons" means, fvwm Icons, fvwm menu icons, MiniIcons which represent applications in various modules, images loaded by modules (e.g., images specified by the *Icon* **FvwmButtons** button option) ...etc. With no arguments this option removes the icon tint.

IconAlpha takes a percentage between 0 and 100 as an argument. It causes fvwm to merge the "icons" which are rendered into the colorset background using this percentage. The default is 100 and it is restored if no argument is given.

Note: It is equivalent to use "Tint a_color rate" and "Alpha a" if a = 100 and the bg color is a_color. This equivalence does not hold for IconAlpha and IconTint as the background can be an image or a gradient (and not a uniform color background). However, in some cases you can achieve (almost) the same effect by using IconTint in the place of IconAlpha. This is preferable as, in general, IconAlpha generates more redrawing than IconTint.

NoShape removes the shape mask from the colorset while *Plain* removes the background pixmap or gradient.

Examples

Colorset 3 fg tan, bg navy

If necessary this creates colorsets 0, 1, 2 and 3 and then changes colorset 3 to have a foreground of tan, a background of navy.

Colorset 3 bg "navy blue"

changes the background color of colorset 3 to navy blue. The foreground and pixmap are unchanged.

Colorset 3 AspectPixmap large_murky_dungeon.xpm

causes depression.

Colorset 3 bg Average

Sets the background color and the relief colors to match the background pixmap. This is the default setting but it must be used if a background color was specified and is now not required.

Colorset 3 YGradient 200 3 blue 1000 navy 1 blue 1000 navy

Adds a Yin Yang gradient background pixmap to colorset 3. If the background is set to average it is recomputed along with the foreground if that is set to contrast.

```
#!/bin/sh
FvwmCommand "Colorset 7 fg navy, bg gray"
while true
do
  FvwmCommand "Colorset 7 fg gray"
  sleep 1
  FvwmCommand "Colorset 7 fg navy"
  sleep 1
done
```

Makes colorset 7 blink.

The color names used in colorsets are saved as fvwm variables which can be substituted in any fvwm command. For example:

```
AddToFunc InitFunction
+ I Exec exec xterm −fg $[fg.cs0] −bg $[bg.cs0]
```

Where $[fg.cs0] is the foreground color of colorset zero. Please refer to the **Command Expansion** section for more information.

CleanupColorsets
Resets a definition of all colorsets.

Color Gradients
A color gradient is a background that changes its color gradually from one hue to a different one. Color gradients can be used by various commands and modules of fvwm. There are eight types of gradients: **HGradient** is a horizontal gradient, **VGradient** is vertical, **DGradient** is diagonal from top left to bottom right, **BGradient** is backwards diagonal from bottom left to top right, **SGradient** is concentric squares, **CGradient** is concentric circles, **RGradient** is a radar like pattern and **YGradient** is a Yin Yang style (but without the dots).

The color gradient syntax has two forms:

?Gradient *colors start−color end−color*

This form specifies a linear gradient. The arguments denote the total number of *colors* to allocate (between 2 and 1000), the initial color and the final color.

Example:

TitleStyle VGradient 20 rgb:b8/ce/bc rgb:5b/85/d0

?Gradient *colors segments color length color* [*length color*] ...

The second form specifies a nonlinear gradient. The arguments are: the total number of *colors* to allocate (between 2 and 1000), then the number of *segments*. For each segment, specify the starting *color*, a relative *length*, then the ending color. Each subsequent segment begins with the second color of the last segment. The lengths may be any non−negative integers. The length of one segment divided by the sum of all segments lengths is the fraction of the colors that are used for the segment.

Examples:

```
MenuStyle * \
        MenuFace DGradient 128 2 lightgrey 50 blue 50 white

# 20% gradient from red to blue,
# 30% from blue to black,
# 50% from black to grey
MenuStyle * \
        MenuFace DGradient 100 3 Red 20 Blue 30 Black 50 Grey

# 50% from blue to green, then
# 50% from yellow to red
Colorset 0 HGradient 128 3 Blue 1000 Green 1 Yellow 1000 Red
```

ENVIRONMENT

The environment variables that have an effect on how fvwm operates are the following:

DISPLAY
Fvwm starts on this display unless the **−display** option is given.

FVWM_MODULEDIR
Set by fvwm to the directory containing the standard fvwm modules.

FVWM_USERDIR
Used to determine the user's data directory for reading and sometimes writing personal files. If this variable is not already set, it is set by fvwm to *$HOME/.fvwm*, which is the default user's data directory.

SESSION_MANAGER
Fvwm tries to contact this session manager.

SESSION_MANAGER_NAME
This is used mainly to determine xsm running to work around its bug. If this variable is set to "xsm", DiscardCommand is set as xsm expects it and not as XSMP requires. If you run fvwm under xsm, you should set this variable to "xsm", otherwise old state files are not removed.

SM_SAVE_DIR
If this is set, fvwm saves its session data in this directory. Otherwise it uses*$HOME*. Note, the state files are named *.fs−??????* and normally are removed automatically when not used anymore.

AUTHORS

Robert Nation with help from many people, based on twm code, which was written by Tom LaStrange. After Robert Nation came Charles Hines, followed by Brady Montz. Currently fvwm is developed by a number of people on the fvwm−workers mailing list.

COPYRIGHT

BUGS

Bug reports can be sent to the fvwm−workers mailing list at <fvwm−workers@fvwm.org>

The official fvwm homepage is *http://fvwm.org/*.