NAME

FvwmButtons − the fvwm buttonbox module

SYNOPSIS

Module FvwmButtons [-g geometry] [-transient | -transientpanel] [name[configfile]]

FvwmButtons can only be invoked by fvwm. Command line invocation of the *FvwmButtons* module will not work.

DESCRIPTION

The FvwmButtons module provides a window of buttons which sits on the X terminal's root window. The user can press the buttons at any time, and trigger invocation of a user-specified command by the window manager. FvwmButtons only works when fvwm is used as the window manager.

The buttonbox can be of any configuration or geometry, and can have monochrome or color icons to represent the actions which would be invoked. Even other applications can be 'swallowed' by the button bar.

Panels that are opened on a button press are available too. See*CREA TING PANELS* section for details.

OPTIONS

The *-g* option specifies the geometry of the main window. The command line option takes precedence over any other geometry settings in the configuration file.

The *-transient* option tells FvwmButtons to terminate itself after the first key or button press has been received (presses to open a sub panel do not count) or a sub panel has been closed or respawned. This is especially useful for sub panels where you want to select a single button and have it closed automatically. It could be used to create two-dimensional graphical menus. Since *-transient* is an option, not a configuration setting you can use the same configuration for transient and non transient button bars.

The *-transientpanel* option does roughly the same as the *-transient* option, but instead of closing the whole button bar, the window is merely hidden. This is very useful if the button bar is started as a subpanel of another button bar because it avoids that it must be started again when something is selected.

INVOCATION

FvwmButtons is spawned by fvwm, so command line invocation will not work.

FvwmButtons can be invoked by inserting the line 'Module FvwmButtons OptionalName' in the .fvwm2rc file. This should be placed in the StartFunction if FvwmButtons is to be spawned during fvwm's initialization. This can be bound to a menu or mouse button or keystroke to invoke it later.

When invoked with the *OptionalName* argument, the *OptionalName* is used to find configuration commands. For example:

AddToFunc StartFunction Module FvwmButtons MyButtonBox

FvwmButtons will then use only the lines starting with "*MyButtonBox", instead of the default "*FvwmButtons".

CONFIGURATION OPTIONS

The following commands are understood by FvwmButtons:

*FvwmButtons: Back *color*
Specifies the background color for the buttons. The relief and shadow color are calculated from the background color.

*FvwmButtons: BoxSize *algorithm*
This option specifies how serious FvwmButtons takes the Rows and Columns options (see below). It can be one of *dumb*, *fixed* or *smart*.

If *fixed* is used and both Rows and Columns are specified and non-zero, FvwmButtons uses exactly the number of rows and columns specified. If the box is too small to accommodate all buttons the module will fail.

If *smart* is used FvwmButtons enlarges the box so all buttons have a chance to fit. The number of columns is increased to at least the width of the widest button and new rows are added until all buttons are placed. For the best tolerance of configuration errors use the smart option.

dumb is neither *fixed* nor *smart.* This is the default.

*FvwmButtons: Colorset *colorset*
Tells the module to use colorset *colorset* for the window background. Refer to the fvwm man page for details about colorsets.

*FvwmButtons: ActiveColorset *colorset*
Tells the module to use colorset *colorset* for the background color/image and/or title color of a button when the mouse is hovering above a button.

*FvwmButtons: PressColorset *colorset*
Tells the module to use colorset *colorset* for the background color/image and/or title color of a button when it is pressed.

*FvwmButtons: Columns *columns*
Specifies the number of columns of buttons to be created. If unspecified, the number of columns is set to the number of buttons requested, divided by the number of rows. If both the rows and columns are specified, but the number of buttons is more than the rows and columns allow for, the columns specification is ignored unless the *BoxSize* option is *fixed*.

*FvwmButtons: File *filename*
Specifies that the configuration for this button is found in the file *filename*. *Filename* can be a full pathname, or is assumed to be in fvwm's startup directory. The configuration file is in the same format as fvwm's configuration file, but each line is read as if prefixed by "*FvwmButtons". Comments are given by starting a line with "#". Line continuation is done by ending a line with a "\".

*FvwmButtons: Font *font*
Specifies the font to be used for labeling the buttons, or *None*.

*FvwmButtons: Fore *color*
Specifies the color used for button label text and monochrome icons.

*FvwmButtons: Frame *width*
Specifies the width of the relief around each button. If this is a negative number, the relief is inverted. This makes the button sunken normally and raised when activated.

*FvwmButtons: Geometry *geometry*
Specifies the FvwmButtons window location and size. The geometry is a standard X11 window geometry specification.

*FvwmButtons: ButtonGeometry *geometry*
This option works like the *Geometry* option except that the size is the size of a single button. The size of the whole FvwmButtons window is calculated by multiplying the button dimension by the number of rows and columns.

*FvwmButtons: Padding *width height*
This option specifies the default horizontal padding to be *width* pixels, and the vertical padding to be *height* pixels. The amount of free space between the relief of the button and its contents is normally 2 pixels on the sides and 4 pixels above and below, except for swallowed windows and containers, which are not padded at all, unless this option is used.

*FvwmButtons: Pixmap *pixmapfile*
Specifies a background pixmap to use. Specify "none" (without the double quotes) for a transparent background.

*FvwmButtons: Rows *rows*
Specifies the number of rows of buttons to be created. The default is 2 rows.

*FvwmButtons: (*options*) [*title icon command*]
Specifies the contents of a button in the buttonbox. The following *options*, separated by commas or whitespace, can be given a button:

geometry
Specifies the size and position of the button within the FvwmButtons window or container. The geometry is a standard X11 window geometry specification. The button is *width* times the normal button width and *height* times the normal button height. If values for *x* and *y* are given, the button is placed x (y) button units from the left (top) of the container if x (y) is positive and x (y) units from the right (bottom) if x (y) is negative. Buttons with position arguments (x and y) are placed before those without them. If two or more buttons are forced to overlap by this, FvwmButtons exits with an error message.

Action [(*options*)] *command*
Specifies an fvwm command to be executed when the button is activated by pressing return or a mouse button. The *command* needs to be quoted if it contains a comma or a closing parenthesis.

The current options of the *Action* are: Mouse *n* - this action is only executed for mouse button *n*. One action can be defined for each mouse button, in addition to the general action.

In the *command* part, you can use a number of predefined variables: *$left*, *$right*, *$top* and *$bottom* are substituted by the left, right, top and bottom coordinates of the button pressed. *$-left*, *$-right*, *$-top* and *$-bottom* are substituted likewise, but the coordinates are calculated from the bottom or the right edge of the screen instead (for a button that is 5 pixels away from the right screen border, $-right will be 5). *$width* and *$height* are replaced by the width or height of the button. The variables *$fg* and *$bg* are replaced with the name of the foreground or background color set with the *Back* or *Fore* option (see below). All this is done regardless of any quoting characters. To get a literal '$' use the string '$$'.

Example:

```
*FvwmButtons: (Title xload, Action (Mouse 1) \
  'Exec exec xload -fg $fg -bg $bg -geometry -3000-3000')
```

Note: With fvwm versions prior to 2.5.0, actions could not be assigned to a button that swallowed an application window (see *Swallow* option). Such actions worked only when the border around the button was clicked. This is now possible, but to get back the old behavior, the *ActionIgnoresClientWindow* can be used on the button:

```
*FvwmButtons: (Action beep, ActionIgnoresClientWindow, \
  Swallow xeyes "Exec exec xeyes")
```

In this example, the action is only executed when you click on the border of the button or the transparent part of the xeyes window, but not on the xeyes window itself.

ActionIgnoresClientWindow
See the note in the description of *Action* above.

ActionOnPress
Usually the action is executed on the button release except for the **Popup** action. This option changes this behavior, the action is executed on the button press. This may be good, for example, with **Menu** or **SendToModule** that generates popups, or when **Frame** is 0 and the button would look unresponsive otherwise.

Back *color*
Specifies the background color to be used drawing this box. A relief color and a shadow color are calculated from this.

Center The contents of the button is centered on the button. This is the default but may be changed by *Left* or *Right*.

Top The contents of the button is vertically aligned at the top of the button. The default is to vertically center it.

Colorset *colorset*
The given colorset can be applied to a container, a swallowed application and a simple button. To apply it to a button or container, simply put the option in a line with a button or container description. Drawing backgrounds for individual buttons and containers with colorsets requires a lot of communication with the X server. So if you are not content with the drawing speed of dozens of buttons with colorset backgrounds, do not use colorsets here. Setting colorsets as the background of swallowed applications does not have this restriction but depends entirely on the swallowed application. It may work as you wish, but since it involves fiddling with other applications' windows there is no guarantee for anything. I have tested three applications: xosview works nicely with a colorset background, xload works only with a VGradient or solid background and an analog xclock leaves a trail painted in the background color after its hands.

If the swallowed window is an fvwm module (see the (No)FvwmModule option to Swallow), then the *colorset* is not applied to the swallowed module. You should use the *colorset* in the module configuration. If the swallowed module has a transparent colorset background, then the FvwmButtons background (and not the button colorset) is seen by transparency of the background of the swallowed module. Refer to the fvwm man page for details about colorsets.

ActiveColorset *colorset*
Use colorset *colorset* for the background color/image and/or title color of the button when the mouse is hovering above it.

PressColorset *colorset*
Use colorset *colorset* for the background color/image and/or title color of the button when it is pressed.

Container [(*options*)]
Specifies that this button will contain a miniature buttonbox, equivalent to swallowing another FvwmButtons module. The options are the same as can be given for a single button, but they affect all the contained buttons. Options available for this use are *Back, Font, Fore, Frame* and *Padding*. Flags for Title and Swallow options can be set with *Title(flags)* and *Swallow(flags)*. You should also specify either "Columns *width*" or "Rows *height*", or "Rows 2" will be assumed. For an example, see the *Sample configuration* section.

The container button itself (separate from the contents) can take format options like *Frame* and *Padding*, and commands can be bound to it. This means you can make a sensitive relief around a container, like

```
*FvwmButtons: (2x2, Frame 5, Padding 2 2, Action Beep,\
  Container(Frame 1))
```

Typically you will want to at least give the container a size setting *width*x*height*.

End Specifies that no more buttons are defined for the current container, and further buttons will be put in the container's parent. This option should be given on a line by itself, i.e

```
*FvwmButtons: (End)
```

Font *fontname*
Specifies that the font *fontname* is to be used for labeling this button.

Fore *color*
Specifies the foregound color of the title and monochrome icons in this button.

Frame *width*
The relief of the button will be *width* pixels wide. If *width* is given as a negative number, the relief is inverted. This makes the button sunken normally and raised when activated.

Icon *filename*
The name of an image file, containing the icon to display on the button. FvwmButtons searches through the path specified in the fvwm ImagePath configuration item to find the

icon file.

ActiveIcon *filename*
The name of an image file, containing an alternative icon to display on the button when the mouse is hovering above the button. If no ActiveIcon is specified, the image specified by Icon is displayed (if there is one).

PressIcon *filename*
The name of an image file, containing an alternative icon to display on the button when the button is pressed. If no PressIcon is specified, the image specified by Icon is displayed (if there is one).

Id *id* The id to be used to identify this button. The first character of the id should be alphabetic. See also the "DYNAMICAL ACTIONS" section.

Left The contents of the button are aligned to the left. The default is to center the contents on the button.

NoSize This option specifies that this button will not be considered at all when making the initial calculations of button sizes. Useful for the odd button that gets just a couple of pixels too large to keep in line, and therefor blows up your whole buttonbox. "NoSize" is equivalent to "Size 0 0".

Padding *width height*
The amount of free space between the relief of the button and its contents is normally 2 pixels to the sides and 4 pixels above and below, except for swallowed windows and containers, which are by default not padded at all. This option sets the horizontal padding to *width* and the vertical padding to *height*.

Panel [(*options*)] *hangon command*
Panels can be swallowed exactly like windows are swallowed by buttons with the *Swallow* command below, but they are not displayed within the button. Instead they are hidden until the user presses the panel's button. Then the panel (the window of the swallowed application) opens with a sliding animation. The *options* can be any of the *flags* described for the Swallow command. In addition a direction 'left', 'right', 'up' or 'down' can be used to specify the sliding direction.

The *steps animation-steps* option defines the number of animation steps.

The *delay ms* option sets the delay between the steps of the animation in milliseconds. Use zero for no delay. The maximum delay is 10 seconds (10000). It doesn't make any sense to use the delay option unless you also use the smooth option.

The *smooth* option causes the panel to redraw between the steps of the animation. The sliding animation may be smoother this way, it depends on the application, and display speed. The application may appear to grow instead of sliding out. The animation may be slower.

The *Hints* option causes FvwmButtons to use the applications size hints to calculate the size of the animation steps. *Hints* is the def ault. If the number of steps is not what you want, try using *NoHints.*

The *noborder* option tells FvwmButtons to ignore the borders of the window when calculating positions for the animation (equivalent to set noplr and noptb in the position option).

With the *indicator* option set, FvwmButtons will draw a small triangle in the button that will open a panel. The triangle points in the direction where the panel will pop up. The *indicator* keyword may be followed by a positive integer that specifies the maximum width and height of the indicator. Without this size FvwmButtons will make the indicator fit the button. You will probably want to use the *Padding* option to leave a few pixels between the indicator and the frame of the button.

The *position* option allows one to place the panel. The syntax is:

position [*context-window*] [*pos*] [*x y*] [*border-opts*]

The argument *context-window* can be one of: Button, Module or Root. The *context-window* is the window from which panel percentage offsets are calculated. Button specifies the panel's button, Module specifies FvwmButtons itself, and Root specifies a virtual screen. The context-window together with the sliding direction define a line segment which is one of the borders of the context-window: the top/bottom/left/right border for sliding up/down/left/right.

The *pos* argument can be one of: center, left or right (for sliding up or a down) or top or bottom (for sliding left or right). It defines the vertical (sliding up and down) or the horizontal (sliding left and right) position of the Panel on the line segment. For example, for a sliding up if you use a left pos, then the left borders of the panel and of the context-window will be aligned.

The offset values *x* and *y* specify how far the panel is moved from it's default position. By default, the numeric value given is interpreted as a percentage of the context window's width (height). A trailing "p" changes the interpretation to mean "pixels". All offset calculations are relative to the buttons location, even when using a root context.

The *border-opts* are: mlr, mtb, noplr and noptb. They define which border widths are taken in account. By default, the borders of FvwmButtons are not taken in account. mlr reverses this default for the left and the right border and mtb reverses this default for the top and the bottom border. Conversely, by default the borders of the Panel are taken in account. noplr reverses this default for the left and the right border and noptb reverses this default for the top and the bottom border.

The defaults are sliding up with a delay of five milliseconds and twelve animation steps. To post the panel without any animation, set the number of steps to zero. The default position is 'Button center'.

Please refer to the *CREATING PANELS* section for further information on panels.

Example:

```
# To include the panel in a button
*FvwmButtons: (Panel(down, delay 0, steps 16) \
  SubPanel "Module FvwmButtons SubPanel")

# To define the panel as an instance of
# FvwmButtons with a different name:
```

```
*SubPanel: (Icon my_lock.xpm, Action Exec xlock)
*SubPanel: (Icon my_move.xpm, Action Move)
...
```

Right The contents of the button are aligned to the right. The default is to center the contents on the button.

Size *width height*
Specifies that the contents of this button require *width* by *height* pixels, regardless of what size FvwmButtons calculates from the icon and the title. A button bar with only swallowed windows will not get very large without this option specified, as FvwmButtons does not consider sizes for swallowing buttons. Note that this option gives the minimum space assured; other buttons might require the buttonbox to use larger sizes.

Swallow [(*flags*)] *hangon command*
Causes FvwmButtons to execute *command*, and when a window with a name, class or resource matching *hangon* appears, it is captured and swallowed into this button. The *hangon* string may contain wildcard characters ('*') that match any substring. Swallow replaces the variables *$fg* and *$bg* as described above for the *Action* option (but if you use the UseOld and NoClose options the application is not be restarted when FvwmButtons is restarted and thus does not get the new colors - if you changed them). An example:

```
*FvwmButtons: (Swallow XClock 'Exec xclock -geometry -3000-3000 &')
```

takes the first window whose name, class, or resource is "XClock" and displays it in the button. If no matching window is found, the "Exec" command creates one. The argument "-geometry -3000-3000" is used so that the window is first drawn out of sight before its swallowed into FvwmButtons.

Modules can be swallowed by specifying the module instead of 'Exec whatever', like:

```
*FvwmButtons: (Swallow "FvwmPager" "FvwmPager 0 0")
```

The flags that can be given to swallow are:

NoClose / Close - Specifies whether the swallowed program in this button will be unswallowed or closed when FvwmButtons exits cleanly. "NoClose" can be combined with "UseOld" to have windows survive a restart of the window manager. The default setting is "Close".

NoHints / Hints - Specifies whether hints from the swallowed program in this button will be ignored or not, useful in forcing a window to resize itself to fit its button. The default value is "Hints".

NoKill / Kill - Specifies whether the swallowed program will be closed by killing it or by sending a message to it. This can be useful in ending programs that doesn't accept window manager protocol. The default value is "NoKill". This has no effect if "NoClose" is specified.

NoRespawn / Respawn / SwallowNew - Specifies whether the swallowed program is to be respawned (restarted) if it dies. If "Respawn" is specified, the program is respawned using the original *command*. Use this option with care, the program might have a

legitimate reason to die. If "SwallowNew" is given, the program is not respawned, but if a new window with the specified name appears, it is swallowed.

NoOld / UseOld - Specifies whether the button will try to swallow an existing window matching the *hangon* name before spawning one itself with *command*. The*hangon* string may contain wildcard characters ('*') that match any substring.The default value is "NoOld". "UseOld" can be combined with "NoKill" to have windows survive a restart of the window manager. If you want FvwmButtons to swallow an old window, and not spawn one itself if failing, let the *command* be "Nop":

```
*FvwmButtons: (Swallow (UseOld) "Console" Nop)
```

If you want to be able to start it yourself, combine it with an action:

```
*FvwmButtons: (Swallow (UseOld) "Console" Nop, \
        Action ‘Exec "Console" console &‘)
```

NoTitle / UseTitle - Specifies whether the title of the button will be taken from the swallowed window's title or not. If "UseTitle" is given, the title on the button changes dynamically to reflect the window name. The default is "NoTitle".

NoFvwmModule / FvwmModule - By default, FvwmButtons treats the swallowed window as an fvwm module window if the 4 first letters of the *command* is "Fvwm" or the 6 first letters of the *command* is "Module". NoFvwmModule and FvwmModule override this logic.

Title [(*options*)] *name*

Specifies the title to be written on the button. Whitespace can be included in the title by quoting it. If a title at any time is too long for its buttons, characters are chopped of one at a time until it fits. If *justify* is "Right", the head is removed, otherwise its tail is removed. These *options* can be given to Title:

Center - The title is centered horizontally. This is the default.

Left - The title is justified to the left side.

Right - The title is justified to the right side.

Side - Causes the title to appear on the right hand side of any icon or swallowed window, instead of below which is the default. If you use small icons, and combine this with the "Left" or "Right" option, you can get a look similar to fvwm's menus.

ActiveTitle *name*

Specifies the title to be written on the button when the mouse is hovering above the button. If no ActiveTitle is specified, the text specified by Title is displayed (if there is any).

PressTitle *name*

Specifies the title to be written on the button when the button is pressed. If no PressTitle is specified, the text specified by Title is displayed (if there is any).

Legacy fields [*title icon command*]

These fields are kept for compatibility with previous versions of FvwmButtons, and their use is discouraged. The *title* field is similar to the option Title *name*. If the title field is

"-", no title is displayed. The *icon* field is similar to the option Icon *filename*. If the icon field is "-" no icon is displayed. The *command* field is similar to the option Action *command* or alternatively Swallow "*hangon*" *command*.

The *command*

Any fvwm command is recognized by FvwmButtons. See fvwm(1) for more information.

The Exec command has a small extension when used in Actions, its syntax is:

```
Exec ["hangon"] command
```

Example:

```
*FvwmButtons: (Action Exec "xload" xload)
```

The hangon string must be enclosed in double quotes. When FvwmButtons finds such an Exec command, the button remains pushed in until a window whose name, class or resource matches the quoted portion of the command is encountered. This is intended to provide visual feedback to the user that the action he has requested will be performed. The hangon string may contain wildcard characters ('*') that match any substring. If the quoted portion contains no characters, then the button will pop out immediately. Note that users can continue pressing the button, and re-executing the command, even when it looks pressed in.

Quoting

Any string which contains whitespace must be quoted. Contrary to earlier versions commands no longer need to be quoted. In this case any quoting character will be passed on to the application untouched. Only commas ',' and closing parentheses ')' have to be quoted inside a command. Quoting can be done with any of the three quotation characters; single quote:

```
'This is a "quote"',
```

double quote:

```
"It's another 'quote'",
```

and back quote:

```
`This is a strange quote`.
```

The back quoting is unusual but used on purpose, if you use a preprocessor like FvwmCpp and want it to get into your commands, like this:

```
#define BG gray60
*FvwmButtons: (Swallow "xload" `Exec xload -bg BG &`)
```

Any single character can be quoted with a preceding backslash '\'.

CREATING PANELS

Former versions of FvwmButtons (fvwm 2.0.46 to 2.3.6) had a different way of handling panels. You can not use your old panel configuration with the new panel feature. Read "CONVERTING OLD PANEL CONFIGURATIONS" for more information.

HOW TO CREATE NEW PANELS

Any program that can be launched from within fvwm and that has a window can be used as a panel. A terminal window could be your panel, or some application like xload or xosview or another fvwm module, including FvwmButtons itself. All you need to know is how to start your application from fvwm.

The button that invokes the panel is as easily configured as any other button. Essentially you need nothing more than the *Panel* option:

```
*FvwmButtons: (Panel my_first_panel \
  "Module FvwmButtons -g -30000-30000 my_first_panel")
*FvwmButtons: (Panel my_second_panel \
  "Exec exec xterm -g -30000-30000 -n my_second_panel")
```

This works like the *Swallow* option. The difference is that the application is not put into the button when it starts up but instead hidden from view. When you press the button for the panel the window slides into view. The '-g -30000-30000' option tells the application that it should be created somewhere very far to the top and left of your visible screen. Otherwise you would see it flashing for a moment when FvwmButtons starts up. Some applications do not work well with this kind of syntax so you may have to live with the short flashing of the window. If you want to make a panel from another instance of FvwmButtons you can do so, but you must give it a different name ('my_first_panel' in above example). If you run FvwmButtons under the same name, new panels are created recursively until your system runs out of resources and FvwmButtons crashes! To configure a second button bar with a different name, simply put '*new_name' in place of '*FvwmButtons' in your configuration file. If you are not familiar with the *Swallow* option or if you want to learn more about how 'swallowing' panels works, refer to the description of the *Swallow* option.

Now that your panel basically works you will want to tune it a bit. You may not want a window title on the panel. To disable the title use the fvwm *Style* command. If your button bar is 'sticky' you may want to make the panel sticky too. And probably the panel window should have no icon in case it is iconified.

```
Style name_of_panel_window NoTitle, Sitcky, NoIcon
```

You may want your panel to stay open only until you select something in it. You can give FvwmButtons the *-transientpanel* option after the -g option in the command. FvwmPager has a similar option '-transient'.

Last, but not least, you can now put an icon, a title or a small arrow in the button so that you can see what it is for. A title or icon can be specified as usual. To activate the arrow, just add '(indicator)' after the 'Panel' keyword in the example above and the *Padding* option to leave a few pixels between the arrow and the border of the button. An optional direction in which the panel is opened can be given too:

```
*FvwmButtons: (Padding 2, Panel(down, indicator) my_first_panel \
  "Module FvwmButtons -g -30000-30000 -transientpanel my_first_panel")
```

There are several more options to configure how your panel works, for example the speed and smoothness of the sliding animation. Please refer to the description of the *Panel* option for further details.

CONVERTING OLD PANEL CONFIGURATIONS

This section describes how to convert a pretty old syntax used in 2.2.x versions. You may skip it if your syntax is more recent.

With the old panel feature you first had one or more lines defining panels in your main FvwmButtons configuration:

```
...
*FvwmButtons(Title WinOps,Panel WinOps)
*FvwmButtons(Title Tools ,Panel Tools)
...
```

After the last configuration line for the main panel the configuration of the first panel followed, introduced with a line beginning with *FvwmButtonsPanel:

```
*FvwmButtonsPanel WinOps
*FvwmButtonsBack bisque2
...

*FvwmButtonsPanel Tools
*FvwmButtonsBack bisque2
...
```

And perhaps you had style commands for you panels:

```
Style FvwmButtonsPanel Title, NoHandles, BorderWidth 0
Style FvwmButtonsPanel NoButton 2, NoButton 4, Sticky
```

The new configuration looks much the same, but now the configuration of the main panel is independent of the configuration of the sub panels. The lines invoking the panels use the same syntax as the Swallow option, so you simply add the name of the window to use as a panel and the command to execute instead of the panel name. Note that you give the new instance of FvwmButtons a different name.

```
*FvwmButtons: (Title WinOps, Panel WinOps \
  "Module FvwmButtons WinOps")
*FvwmButtons: (Title Tools , Panel Tools \
  "Module FvwmButtons Tools")
```

If you used something like ’Panel-d’ you now have to use ’Panel(down)’ instead. To make the new panel vanish as soon as a button was selected start FvwmButtons with the ’-transientpanel’ option:

```
*FvwmButtons: (Title Tools , Panel(down) Tools \
  "Module FvwmButtons -transientpanel Tools")
```

The rest of the configuration is very easy to change. Delete the lines '*FvwmButtonsPanel <name>' and add <name> to all of the following configuration lines for the panel instead. Use the same name in your Style commands:

```
*WinOps: Back bisque2
...
*Tools: Back bisque2
...
Style "WinOps" Title, NoHandles, BorderWidth 0
Style "WinOps" NoButton 2, NoButton 4, Sticky
Style "Tools" Title, NoHandles, BorderWidth 0
Style "Tools" NoButton 2, NoButton 4, Sticky
```

That's it. The new panels are much more flexible. Please refer to other parts of this documentation for details.

WHY WAS THE PANEL FEATURE REWRITTEN?

There are several reasons. The most important one is that the program code implementing the panels was very disruptive and caused a lot of problems. At the same time it made writing new features for FvwmButtons difficult at best. The second reason is that most users were simply unable to make it work - it was way too complicated. Even I (the author of the new code) had to spend several hours before I got it working the first time. The third reason is that the new panels are more versatile. Any application can be a panel in FvwmButtons, not just other instances of FvwmButtons itself. So I sincerely hope that nobody is angry about the change. Yes - you have to change your configuration, but the new feature is much easier to configure, especially if you already know how the Swallow option works.

ARRANGEMENT ALGORITHM

FvwmButtons tries to arrange its buttons as best it can, by using recursively, on each container including the buttonbox itself, the following algorithm.

Getting the size right

First it calculates the number of button unit areas it will need, by adding the width times the height in buttons of each button. Containers are for the moment considered a normal button. Then it considers the given *rows* and *columns* arguments. If the number of rows is given, it will calculate how many columns are needed, and stick to that, unless *columns* is larger, in which case you will get some empty space at the bottom of the buttonbox. If the number of columns is given, it calculates how many rows it needs to fit all the buttons. If neither is given, it assumes you want two rows, and finds the number of columns from that. If the BoxSize option is set to *smart* at least the height/width of the tallest/widest button is used while the *fixed* value prevents the box from getting resized if both *rows* and *columns* have been set to non-zero.

Shuffling buttons

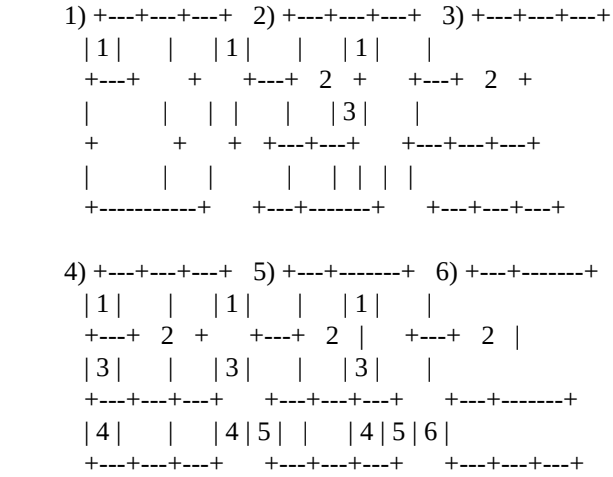
Now it has a large enough area to place the buttons in, all that is left is to place them right. There are two kinds of buttons: fixed and floating buttons. A fixed button is forced to a specific slot in the button box by a x/y geometry argument. All other buttons are considered floating. Fixed buttons are placed first. Should a fixed button overlap another one or shall be place outside the buttons window, FvwmButtons exits with an error message. After that the floating buttons are placed. The algorithm tries to place the buttons in a left to right, top to bottom western fashion. If a button fits at the suggested position it is placed there, if not the current slot stays empty and the slot to the right will be considered. After the button has been placed, the next button is tried to be placed in the next slot and so on until all buttons are placed. Additional rows are added below the bottom line of buttons until all buttons are placed if necessary if the BoxSize option *smart* is used.

Containers

Containers are arranged by the same algorithm, in fact they are shuffled recursively as the algorithm finds them.

Clarifying example

An example might be useful here: Suppose you have 6 buttons, all unit sized except number two, which is 2x2. This makes for 5 times 1 plus 1 times 4 equals 9 unit buttons total area. Assume you have requested 3 columns.



What size will the buttons be?

When FvwmButtons has read the icons and fonts that are required by its configuration, it can find out which size is needed for every non-swallowing button. The unit button size of a container is set to be large enough to hold the largest button in it without squeezing it. Swallowed windows are simply expected to be comfortable with the button size they get from this scheme. If a particular configuration requires more space for a swallowed window, it can be set in that button's configuration line using the option "Size *width height*". This will tell FvwmButtons to give this button at least *width* by *height* pixels inside the relief and padding.

DYNAMICAL ACTIONS

A running FvwmButtons instance may receive some commands at run time. This is achieved using the fvwm command

SendToModule FvwmButtons-Alias <action> <params>

Supported actions:

ChangeButton *button_id* options

can be used to change the title or icon of a button at run time. *b utton_id* is the id of the button to change as specified using the **Id** button option. It may also be a number, in this case the button with the given number is assumed. And finally, *button_id* may be in the form +x+y, where x and y are a column number and a row number of the button to be changed. It is possible to specify multiple option pairs (name with value) by delimiting them using comma. Currently options include **Title**, **ActiveTitle**, **PressTitle**, **Colorset**, **Icon**, **ActiveIcon** and **PressIcon**. These options work like the configuration options of the same name.

ExpandButtonVars *button_id* command
replaces variables present in the *command* exactly like in the **Action** button option and then sends the command back to fvwm. *button_id* has the same syntax as described in **ChangeButton** above.

PressButton *button_id* [*mouse_button*]
simulates a mouse click on a button. *button_id* is the id of the button to press as specified using the **Id** button option and *mouse_button* is the number of mouse button used to click on the button e.g "1" for the left mouse button etc. Quotes around the number are not necessary. If*mouse_b utton* option is omitted, mouse button 1 is assumed. This command behaves exactly as if the mouse button was pressed and released on the button on in question.

Silent This prefix may be specified before other actions. It disables all possible error and warning messages.

Example:

```
*FvwmButtons: (Id note1, Title "13:30 - Dinner", Icon clock1.xpm)

SendToModule FvwmButtons Silent \
  ChangeButton note1 Icon clock2.xpm, Title "18:00 - Go Home"
```

SAMPLE CONFIGURATION

The following are excerpts from a .fvwm2rc file which describe FvwmButtons initialization commands:

```
############################################################
# Load any modules which should be started during fvwm
# initialization

# Make sure FvwmButtons is always there.
AddToFunc StartFunction  "I" Module FvwmButtons

# Make it titlebar-less, sticky, and give it an icon
Style "FvwmButtons"        Icon toolbox.xpm, NoTitle, Sticky

# Make the menu/panel look like CDE
Style "WinOps" Title, NoHandles, BorderWidth 0
Style "WinOps" NoButton 2, NoButton 4, Sticky
Style "Tools" Title, NoHandles, BorderWidth 0
Style "Tools" NoButton 2, NoButton 4, Sticky

############################################################
DestroyModuleConfig FvwmButtons: *
*FvwmButtons: Fore Black
*FvwmButtons: Back rgb:90/80/90
*FvwmButtons: Geometry -135-5
*FvwmButtons: Rows 1
*FvwmButtons: BoxSize smart
*FvwmButtons: Font -*-helvetica-medium-r-*-*-12-*
*FvwmButtons: Padding 2 2

*FvwmButtons: (Title WinOps, Panel WinOps \
```

```
  "Module FvwmButtons -transientpanel WinOps")
*FvwmButtons: (Title Tools, Panel Tools   \
  "Module FvwmButtons -transientpanel Tools")

*FvwmButtons: (Title Resize, Icon resize.xpm,  Action Resize)
*FvwmButtons: (Title Move,   Icon arrows2.xpm, Action Move  )
*FvwmButtons: (Title Lower,  Icon Down,        Action Lower )
*FvwmButtons: (Title Raise,  Icon Up,          Action Raise )
*FvwmButtons: (Title Kill,   Icon bomb.xpm,    Action Destroy)

*FvwmButtons: (1x1,Container(Rows 3,Frame 1))
*FvwmButtons: (Title Dopey ,Action                        \
    'Exec "big_win" xterm -T big_win -geometry 80x50 &')
*FvwmButtons: (Title Snoopy, Font fixed, Action           \
    'Exec "small_win" xterm -T small_win &')
*FvwmButtons: (Title Smokin')
*FvwmButtons: (End)

*FvwmButtons: (Title Xcalc, Icon rcalc.xpm,               \
          Action 'Exec "Calculator" xcalc &')
*FvwmButtons: (Title XMag, Icon magnifying_glass2.xpm,    \
          Action 'Exec "xmag" xmag &')
*FvwmButtons: (Title Mail, Icon mail2.xpm,                \
          Action 'Exec "xmh" xmh &')
*FvwmButtons: (4x1, Swallow "FvwmPager" 'FvwmPager 0 3'   \
          Frame 3)

*FvwmButtons: (Swallow(UseOld,NoKill) "xload15" 'Exec xload \
    -title xload15 -nolabel -bg rgb:90/80/90 -update 15   \
    -geometry -3000-3000 &')
```

The last lines are a little tricky - one spawns an FvwmPager module, and captures it to display in a quadruple width button. is used, the Pager will be as big as possible within the button's relief.

The final line is even more magic. Note the combination of *UseOld* and *NoKill*, which will try to swallow an existing window with the name "xload15" when starting up (if failing: starting one with the specified command), which is un-swallowed when ending FvwmButtons. The swallowed application is started with "-geometry -3000-3000" so that it will not be visible until its swallowed.

The other panels are specified after the root panel:

```
########## PANEL WinOps
DestroyModuleConfig WinOps: *
*WinOps: Back bisque2
*WinOps: Geometry -3-3
*WinOps: Columns 1

*WinOps: (Title Resize, Icon resize.xpm,  Action Resize)
*WinOps: (Title Move,   Icon arrows2.xpm, Action Move  )
*WinOps: (Title Lower,  Icon Down,        Action Lower )
*WinOps: (Title Raise,  Icon Up,          Action Raise )
```

```
########## PANEL Tools
DestroyModuleConfig Tools: *
*Tools: Back bisque2
*Tools: Geometry -1-1
*Tools: Columns 1

*Tools: (Title Kill,    Icon bomb.xpm,    Action Destroy)
```

The color specification *rgb:90/80/90* is actually the most correct way of specifying independent colors in X, and should be used instead of the older *#908090*. If the latter specification is used in your configuration file, you should be sure to escape the hash in any of the *command*s which will be executed, or fvwm will consider the rest of the line a comment.

Note that with the x/y geometry specs you can easily build button windows with gaps. Here is another example. You can not accomplish this without geometry specs for the buttons:

```
############################################################
# Another example
############################################################

# Make it titlebar-less, sticky, and give it an icon
Style "FvwmButtons"        Icon toolbox.xpm, NoTitle, Sticky

DestroyModuleConfig FvwmButtons: *
*FvwmButtons: Font       5x7
*FvwmButtons: Back rgb:90/80/90
*FvwmButtons: Fore       black
*FvwmButtons: Frame      1
# 9x11 pixels per button, 4x4 pixels for the frame
*FvwmButtons: Geometry    580x59+0-0
*FvwmButtons: Rows       5
*FvwmButtons: Columns     64
*FvwmButtons: BoxSize     fixed
*FvwmButtons: Padding     1 1

# Pop up a module menu directly above the button.
*FvwmButtons: (9x1+3+0, Padding 0, Title "Modules",   \
  Action ‘Menu Modulepopup rectangle \
  $widthx$height+$lleft+$top o+50 -100m‘)

# first row of buttons from left to right:
*FvwmButtons: (3x2+0+1, Icon my_lock.xpm, Action ‘Exec xlock‘)
*FvwmButtons: (3x2+3+1, Icon my_recapture.xpm, Action Recapture)
*FvwmButtons: (3x2+6+1, Icon my_resize.xpm, Action Resize)
*FvwmButtons: (3x2+9+1, Icon my_move.xpm, Action Move)
*FvwmButtons: (3x2+12+1, Icon my_fvwmconsole.xpm,     \
  Action ’Module FvwmConsole’)

# second row of buttons from left to right:
*FvwmButtons: (3x2+0+3, Icon my_exit.xpm, Action QuitSave)
*FvwmButtons: (3x2+3+3, Icon my_restart.xpm, Action Restart)
*FvwmButtons: (3x2+6+3, Icon my_kill.xpm, Action Destroy)
*FvwmButtons: (3x2+9+3, Icon my_shell.xpm, Action ’Exec rxvt’)
```

```
# big items
*FvwmButtons: (10x5, Swallow (NoKill, NoCLose)       \
  "FvwmPager" 'FvwmPager * * -geometry 40x40-1024-1024')
*FvwmButtons: (6x5, Swallow "FvwmXclock" `Exec xclock \
  -name FvwmXclock -geometry 40x40+0-3000 -padding 1  \
  -analog -chime -bg rgb:90/80/90`)
*FvwmButtons: (13x5, Swallow (NoClose)              \
"FvwmIconMan" 'Module FvwmIconMan')
*FvwmButtons: (20x5, Padding 0, Swallow "xosview"    \
  `Exec /usr/X11R6/bin/xosview -cpu -int -page -net  \
  -geometry 100x50+0-3000 -font 5x7`)
```

BUGS

The action part of the Swallow option must be quoted if it contains any whitespace character.

COPYRIGHTS

The FvwmButtons program, and the concept for interfacing this module to the Window Manager, are all original work by Robert Nation.

Further modifications and patching by Jarl Totland, copyright 1996. The statement above still applies.

AUTHOR

Robert Nation. Somewhat enhanced by Jarl Totland, Jui-Hsuan Joshua Feng, Scott Smedley.