Chapter 1. An Introduction to Pekwm

The Pekwm Window Manager is written by Claes Nästén. The code is based on the aewm++ window manager, but it has evolved enough that it no longer resembles aewm++ at all. It also has an expanded feature-set, including window grouping (similar to ion, pwm, or fluxbox), auto properties, xinerama and keygrabber that supports keychains, and much more.

Why Pekwm?

"Why make another window manager?", some ask. This may confuse some people, but the best answer is "Why not?". There are arguments out there that it's better to have a single standard desktop environment, so that our mothers can find their way around, but in all honestly, if most of us wanted the same environment as our mothers, we probably wouldn't be reading this anyway. The same can also be applied to Your sister, your roommate, your wife, even your cat.

"Why should I use pekwm?", others ask. Nobody ever said you should. However, we use it. And you're welcome to as well. You should use the environment most suited to you. For a better answer to this question, Check out the Pekwm Features section below.

Pekwm Features

Here's a short list of some of the features included in pekwm:

- Possibility to group windows in a single frame
- Configurable keygrabber that supports keychains
- Configurable mouse actions
- Configurable root- and window-menus and keybindings for all menus
- Dynamic menus that regenerate on every view from a script output
- Multi-screen support both via RandR and Xinerama
- Configurable window placement
- Theming support with images, shaping and configurable buttons.
- Autoproperties (Automatic properties such as a window's sticky state, etc.)

Chapter 2. Getting Pekwm

Now that you've decided to try it out, you need to get it. You're left with two options. The first is to download and compile the source, and the second is finding a pre-compiled package.

Getting the Pekwm source

The source code is available from the pekwm website, http://pekwm.org.

Files are named pekwm-0.1.12.tar.gz and pekwm-0.1.12.tar.bz2. Although it doesn't matter which you get, keep in mind that the .bz2 is smaller.

Getting prebuilt Pekwm packages

Links to pre-built pekwm packages are available at the pekwm website, http://pekwm.org.

The current version of pekwm is 0.1.12.

If there's no package for your distribution, and you'd like to build one, Let us know! We'll gladly host or link to binary packages.

Chapter 3. Compiling Pekwm

This chapter will help you get pekwm compiled.

Unpacking the Archive

The first step to compiling pekwm is to unpack the archive. Unpacking it depends on which version you downloaded:

```
tar -zxvf pekwm-0.1.12.tar.gz
tar -jxvf pekwm-0.1.12.tar.bz2
```

> **Note:** The '-j' option works normally on most linux systems, and as of the current GNU tar development version, is part of GNU tar. If your system does not support the -j option, you can use two things: **bzip2 -dc pekwm-0.1.12.tar.bz2 | tar -xvf -** or **bzip2 -d pekwm-0.1.12.tar.bz2** followed by **tar -xvf pekwm-0.1.12.tar**. This also works for the .tar.gz version using **gzip -dc** or **gzip -d**.

The 'v' options are optional, they show you the filenames as they're being extracted. at this point, you should have a pekwm-0.1.12 directory. Use **cd pekwm-0.1.12** to get there.

Configuration Options

The first thing to do is to run the configure script. This configures compile options for pekwm. Here are some of the more used options and what their default values are.

Important ./configure options:

--enable-shape

> Enables the use of the Xshape extension for non-rectangular windows.
>
> By default, Enabled

--enable-xinerama

> Enables Xinerama multi screen support
>
> By default, Enabled

--enable-xrandr

> Enables RandR multi screen support
>
> By default, Enabled

--enable-xft

> Enables Xft font support in pekwm (themes).
>
> By default, Enabled

--enable-image-xpm

> XPM image support using libXpm.

> By default, Enabled

--enable-image-jpeg

> JPEG image support using libjpeg.

> By default, Enabled

--enable-image-png

> PNG image support using libpng.

> By default, Enabled

--enable-debug

> Enables debugging output

> By default, Disabled

--enable-pedantic

> Enables pedantic compile flags when using GCC

> By default, Disabled

--enable-menus

> Enables menu support (It's useful to turn this off if you rely completely on a desktop environment using the root window)

> By default, Enabled

--enable-harbour

> Enables the use of the harbour which is used to swallow Window Maker dockapps.

> By default, Enabled

--prefix=PREFIX

> It may be useful to use a custom prefix to install the files.

> By default, /usr/local

Building and installing

After running ./configure with any options you need, run **make**. This should only take a few minutes. After that, become root (unless you used a prefix in your home directory, such as --prefix=/home/you/pkg) and type **make install**

Adding **exec pekwm** to ~/.xinitrc if you start X running **startx** or ~/.xsession if you use a display manager should usually be enough to get pekwm running.

That's it! pekwm is installed on your computer now. Next you should read the Getting Started chapter.

Chapter 4. Getting Started

Now that you have pekwm installed, you should take a few moments to test how the basics work.

The documentation generally tries to explain the terms it uses, but useful terms to know beforehand include "Mod1" which usually means your Alt key, and "Mod4" which refers to the "windows key" found on recent keyboards.

It's also good to know that a "frame" basically means the same as a window, but this window can contain one or more real windows. The same concept is also referred as a "window group". In relation to this, window inside such a frame can be referred as a "grouped window" or a "client window", or simply just as a "client".

First Run

The first time you run pekwm, the following files should be copied into your `~/.pekwm` directory: `config`, `menu`, `keys`, `autoproperties`, `start`, `vars`. You will learn more about these files in the Configuration section.

All this happens behind-the-scenes, so by default, you'll be placed into a nice working environment.

About Menus and Iconification

When you iconify (This is the traditional name in unix. Windows calls this minimizing.) a window in pekwm, it doesn't really go anywhere like you might expect. You can de-iconify using one of three menus: The Icon menu, the Goto menu, or the GotoClient menu. When you click on an item in one of these menus, it takes you to that window deiconifying when necessary.

Icon menu shows you a list of all currently iconified windows. Use Mod4+Shift+I to bring it up.

Goto menu shows you a list of windows currently active. This menu will only show the currently active window of possible window groups. Use Mod4+L, or middle click of the mouse on the root window or screen edges to bring this menu up.

GotoClient menu shows you a list of every window currently open. Window groups are separated from each other with a menu separator which is defined by the currently used theme, usually a line of some sort. You need to be using window grouping to really see any difference between the GotoClient and Goto menus. Use Mod4+C, or middle click of the mouse on the root window or screen edges while holding down Mod4 to bring up the GotoClient menu.

An item in the goto and gotoclient menu and icon menu (and attach menus) has the following syntax:

```
<number> [symbols] Window title
```

The number represents what workspace the window is on. Symbols is a list of symbols that represent window states. They are:

Symbols in the pekwm menus

- * (sticky)
- . (iconified)

- ^ (shaded)

- + (above normal window layer)

- - (below normal window layer)

- A (active in group)

> **Note:** If you are using window grouping, the whole group will iconify instead of one window. Please ungroup before minimizing if you wish to iconify a single client window from a group frame.

Using the mouse

Pekwm has excellent mouse support. Here you'll learn how to do some usual window management actions using the default configuration.

Moving windows is rather easy and I think you already got the hang of using the left mouse button on the titlebar and dragging. But did you notice that when you press Mod1 while dragging on the client window (not the titlebar) it works just as well.

Resizing is also easy and most are familiar with it. Hang on to a border of the window with the left mouse button and drag. Release the button and you're done. But what you likely didn't know is that if you press Mod1 and then drag on the client window with the right mouse button it also makes windows resize. Try it, it's great.

Minimizing (iconifying) with the mouse is possible thru the window menu. Right click on a windows titlebar and select Iconify. Many themes also implement a iconifying button on the titlebar. Also see About Menus and Iconification.

Shading is done by double clicking the titlebar with the middle mouse button. Unshade doing it again.

Maximizing is quite easy. Hold Mod1 and double click with the left button. Many themes also have a maximize button in them. The default theme has one on the right corner of the titlebar. It's also possible to use the window menu (right click on titlebar).

Filling. Sounds odd? Its not. It just means you can make a window grow as large as it can until it hits the borders of the windows surrounding it. Easy as pie, double click on the titlebar with the left mouse button. Excellent feature you are likely to grow to like.

Raising windows. Easy. Left click on the windows titlebar or hold Mod1 and left click anywhere on the window.

Lowering windows. Hold Mod4 and left click anywhere on the window.

Closing. Most themes implement a close button. Default theme has one on the left end of the titlebar. You can also close a client by holding Mod4 and right clicking on its title. Note that the client doesn't have to be the active client of the frame for this to work. Also see the window menu.

Grouping. Middle click and drag on a titlebar and release over the frame you want the window into. Holding Mod1 and middle clicking works on the whole client window. This process can also be automated, more on that later.

Activating clients. Now that you have multiple clients grouped into one frame, you can switch between them simply by left clicking on the clients title. Doing so also raises the frame. If you don't want the frame to raise, middle click on the clients title. Also try turning the mouse wheel on a frames titlebar when it has more than one client.

Menus. As mentioned, press the right mouse button on a windows titlebar and you get the window menu. You can do lots of things from there that are not possible by mouse shortcuts. To bring up the root menu (the one you use to launch programs) click the right button on the background or on the screen edges. To get the Goto menu, click the middle button on the background or screen edges.

Most theme buttons work with a left click. Some also have specials when you use other mouse buttons on them. Like the default themes maximize button. Try it. The default themes close button also has a special when you right click on it. With it it is possible to kill the client if it's so stuck you can't close it normally.

That ends our short introduction to using the mouse in pekwm. Hope you found the defaults pleasant to use. Remember that if you didn't like something, you can change it. See Mouse Bindings for how.

Using the keyboard

Pekwm allows excellent keyboard control of your window management. Lets try it out a bit. If you don't have the windows key on your keyboard, please see `~/.pekwm/keys` for the keychains you can use to do the same and a lot more.

Moving and Resizing windows. To be able to move and resize windows you have to activate the special MoveResize state. This happens by pressing Mod4+Enter. The window should after this be movable by using the arrow keys. To resize, press Mod4 and use the arrows. Using the Shift-key with these actions makes them be careful. To accept the new size and position, press Enter. To fail back to the old position and size press Escape.

Minimizing. Press Mod4+I. Mod4+Shift+I pops up the icon menu you can use to bring it back.

Shading. This is to hide most of the window, leaving only the titlebar visible. Press Mod4+S to toggle the shaded state.

Maximizing. Mod4+M toggles the maximized state.

Filling (making a window grow as big as it can in the space it has around it). Press Mod4+G to make windows grow to fit.

Fullscreen. Press Mod4+F to toggle the fullscreen state.

Moving between frames. Press Mod1+Tab and Mod1+Shift+Tab to move between frames. Or use Mod1+Ctrl+Tab and Mod1+Ctrl+Shift+Tab to move between most recently used frames. You can also use directional focusing. Press Mod4 and one of the arrow keys. The focus should change to the frame that is in the direction you pointed to. Try it out.

Moving inside frames. Press Mod4+Tab and Mod4+Shift+Tab to move between the clients in a frame.

Closing. Press Mod4+Q to close windows.

Grouping. The easiest way to group is to use marking. You select clients you want to group to another frame by toggling them marked with Mod4+Z. You can have as many marked clients as you wish. Then go to the frame you want those now marked clients to be attached and press Mod4+A. That's it.

Menus. There are some simple menu bindings. Mod4+R shows your main menu (the Root menu). Mod4+L shows a list of your active windows (the Goto menu). Mod4+C shows a list of all your open windows (the Goto menu). Mod4+W brings up the Window menu. And Mod4+Shift+I the Icon menu.

Those were the basics. There's a ton more. See the rest of the documentation for rest of the simple bindings and `~/.pekwm/keys` for a list of the keychains. And again, if you hated something, go ahead and edit it.

Chapter 5. Window Grouping

The main feature of pekwm is window grouping, similar to that of ion, or the tabs in pwm or fluxbox.

What is window grouping?

Window grouping is a very simple concept, but it could be hard to understand at first. It's a simple way of making multiple applications share the exact same space.

The simplest way to explain this is with an analogy. Imagine you have 20 sheets of paper. To save space, you stack them on top of each other. then, you have little tabs sticking out of one edge so you can quickly flip to any sheet of paper.

You have likely stumbled upon a WWW-browser that calls this tabbing. In pekwm, Window grouping is visually done by dividing up the physical space of the titlebar. We don't call them tabs for historical reasons, but refer to them as "clients". Windows that can contain any number of clients are more than often referred as "frames".

Also note that a frame can contain any type of clients. If you want to group one of your WWW-browser windows with your text editor for future reference, you're free to do so.

How window grouping works

The first thing to know is how to group one window to another. Middle-Click (On a normal X setup, the 2nd mouse button is the middle button) the titlebar of the first window and begin to drag the window. You should now see a rectangle with the window's title in it. Drag that rectangle to above the target window, and release your mouse button.

> **Note:** Any time this document mentions a key or mouse button, there's a strong likelihood that you can change which key or mouse button is used for that function. Please see the Keyboard and Mouse config section.

Now that you have windows in a group, you need to learn to choose between windows in that group. The first way is by clicking the middle mouse button on the window's part of the titlebar. That window should now be the currently-active window of the group. You can also use a keybinding for this. The default keybindings are Mod4+Tab and Mod4+Shift+Tab to go forward and back between active window in the frame.

To de-group, simply middle click and drag the window off the frame, and release anywhere. If you release over another window group, you'll move the window to the new group. Default keybinding for detaching clients from a group is first Ctrl+Mod1+T then D.

You can also set windows up to automatically be grouped to one another. See the Autoproperties section for more details.

Advanced Grouping Topics

Another thing you can do with window grouping is Tagging. This is done by setting the toggleable attribute "tagged" on a frame with the action "Set Tagged". A tag is like a miniature autogroup. It says "All new windows launched should be automatically grouped to this Frame" and all other autogrouping defined in the autoproperties will be ignored while it is set. "UnSet Tagged" removes the tag. Default keybinding to toggle tagging on a frame is Ctrl+Mod1+T then T. Unsetting tagging works even if the window you have set tagged isn't active. It's default keybinding is Ctrl+Mod1+T then C.

You can toggle all autogrouping on and off with the toggleable attribute GlobalGrouping. To disable you need to use the action "Unset GlobalGrouping" and to enable autogrouping use "Set GlobalGrouping". The default keybinding that toggless between set and unset is Ctrl+Mod1+T then G.

You can set a marked state on clients with "set marked" and then attach those marked clients to another frame by focusing the frame you want the marked clients attached to and then using the AttachMarked action. By default marking can be reached with two simple keybindings. Mod4+Z toggles a clients Marked state and Mod4+A attaches clients with marked state set into the current frame. Marked clients will have "[M]" appended to their titlebars.

Pekwm also includes some menus that have to do with grouping. AttachClientInFrame (Ctrl+Mod1+M, A) sends the current client to the selected frame. AttachFrameInFrame (Ctrl+Mod1+M, F) sends the contents of the current frame to the selected frame. AttachClient (Ctrl+Mod1+M, Shift+A) brings the selected client into the current frame. AttachFrame (Ctrl+Mod1+M, Shift+F) brings the contents of the selected frame into the current frame.

Chapter 6. Workspaces

Workspaces in pekwm are a very common feature, found in almost every UNIX window manager in existence. They're also called desktops in some window managers. In Pekwm-Speak, "workspace", "desktop", and "desk" are interchangeable. Use whichever one you feel like using.

> **Note:** By default, pekwm enables four workspaces. You can change this by editing your `~/.pekwm/config` file. See The main config file section for more details.

Workspace Navigation

You can send windows to another workspace by right-clicking the titlebar, going to 'send to' and picking the desktop you'd like. Another option is using the SendToWorkspace keybindings (by default, Mod4 and one of F1, F2, F3, or F4). Using the mouse to drag a window over the right or left screen edge makes it move to the next or previous workspace. Also try placing the mouse pointer on a client window and rotating the mouse wheel while holding Mod1 down to send a window to the next or previous workspace and follow it there yourself.

Switch desktops by using the GoToWorkspace keybindings (by default Mod4 and one of 1,2,3, or 4), or the "GotoWorkspace Next" and "GotoWorkspace Prev" actions (by default Ctrl+Mod1+Right and Ctrl+Mod1+Left). Holding Mod1 key while moving the mouse pointer over the right or left screen edge will make you move to the next or previous workspace. Also pressing the left mouse button on the right or left screen edge will make you move to the next or previous workspace. Using the mouse wheel on the background or the screen edges also changes your workspace.

Chapter 7. The Pekwm Common Syntax for Config Files

Basic Syntax

All pekwm config files (with the exception of the start file- see start file) follow a common syntax for options.

```
# comment
// another comment
/*
 yet another comment
*/

$VARIABLE = "Value"
$_ENVIRONMENT_VARIABLE = "Value"
INCLUDE = "another_configuration.cfg"
COMMAND = "program to execute and add the valid config syntax it outputs here"

# Normal format
Section = "Name" {
 Event = "Param" {
  Actions = "action parameter; action parameter; $VAR $_VARIABLE"
 }
}

// Compressed format
Section = "Name" { Event = "Param" { Actions = "action parameters; action parameters; $VAR
```

You can usually modify the spacing and line breaks, but this is the "Correct" format, so the documentation will try to stick to it.

Events can be combined into the same line by issuing a semicolon between them. Actions can be combined into the same user action by issuing a semicolon between them. You can use an INCLUDE anywhere in the file.

Pekwm has a `vars` file to set common variables between config files. Variables are defined in `vars` and the file is INCLUDEd from the configuration files.

Comments are allowed in all config files, by starting a comment line with # or //, or enclosing the comments inside /* and */.

Template Syntax

The pekwm configuration parser supports the use of templates to reduce typing. Template support is currently available in autoproperties and theme configuration files. Template configuration is not fully compatible with the non-template syntax and thus requires activation to not break backwards compatibility. To enable template parsing start the configuration file with the following:

```
Require {
 Templates = "True"
}
```

Defining templates is just like creating an ordinary section, but uses the special name Define. Below is an example defining a template named NAME:

```
Define = "NAME" {
  Section = "Sub" {
    ValueSub = "Sub Value"
  }
  Value = "Value"
}
```

The above defined template can later be used by using the magic character @. The below example shows usage of the template NAME in a two sections named Name and NameOverride overriding one of the template values:

```
Section = "Name" {
  @NAME
}
Section = "NameOverride" {
  @NAME
  Value = "Overridden value"
}
```

The above example is equivalent of writing the following:

```
Section = "Name" {
  Section = "Sub" {
    ValueSub = "Sub Value"
  }
  Value = "Value"
}
Section = "Name" {
  Section = "Sub" {
    ValueSub = "Sub Value"
  }
  Value = "Overridden"
}
```

Variables In Pekwm Config Files

Pekwm config enables you to use both internal to pekwm variables, as well as global system variables. Internal variables are prefixed with a **$**, global variables with **$_**.

```
# examples of how to set both type of variables
$INTERNAL = "this is an internal variable"
$_GLOBAL = "this is a global variable"

# examples of how to read both type of variables
RootMenu = "Menu" {
```

```
 Entry = "$_GLOBAL" { Actions = "xmessage $INTERNAL" }
}
```

There is one special global variable pekwm handles. It is called $_PEKWM_CONFIG_FILE. This global variable is read when pekwm starts, and it's contents will be used as the default config file. It will also be updated to point to the currently active config file if needed.

Variables can probably be defined almost anywhere, but it's probably a better idea to place them at the top of the file, outside of any sections.

Chapter 8. The main config file

The main config file is where all the base config stuff goes.

Basic Config

As previously indicated, the config file follows the rules defined in Common Syntax.

Here's an example `~/.pekwm/config` file:

```
Files {
 Keys = "~/.pekwm/keys"
 Mouse = "~/.pekwm/mouse"
 Menu = "~/.pekwm/menu"
 Start = "~/.pekwm/start"
 AutoProps = "~/.pekwm/autoproperties"
 Theme = "~/.pekwm/themes/default"
 Icons = "~/.pekwm/icons/"
}

MoveResize {
 EdgeAttract = "10"
 EdgeResist = "10"
 WindowAttract = "5"
 WindowResist = "5"
 OpaqueMove = "True"
 OpaqueResize = "False"
}

Screen {
 Workspaces = "4"
 WorkspacesPerRow = "2"
 WorkspaceNames = "Main;Web;E-mail;Music"
 ShowFrameList = "True"
 ShowStatusWindow = "True"
 ShowStatusWindowCenteredOnRoot = "False"
 ShowClientID = "False"
 ShowWorkspaceIndicator = "500"
 FocusNew = "True"
 PlaceNew = "True"

 TrimTitle = "..."
 FullscreenAbove = "True"
 FullscreenDetect = "True"
 HonourRandr = "True"
 HonourAspectRatio = "True"
 EdgeSize = "1 1 1 1"
 EdgeIndent = "False"
 PixmapCacheSize = "20"
 DoubleClickTime = "250"

 Placement {
```

```
  Model = "CenteredOnParent Smart MouseCentered"
  Smart {
   Row = "False"
   TopToBottom = "True"
   LeftToRight = "True"
   OffsetX = "0"
   OffsetY = "0"
  }
 }

 UniqueNames {
  SetUnique = "True"
  Pre = " #"
  Post = ""
 }
}

Menu {
 DisplayIcons = "True"

 Icons = "DEFAULT" {
  Minimum = "16x16"
  Maximum = "16x16"
 }

 Select = "Motion"
 Enter = "Motion ButtonPress"
 Exec = "ButtonRelease"
}

CmdDialog {
 HistoryUnique = "True"
 HistorySize = "1024"
 HistoryFile = "~/.pekwm/history"
 HistorySaveInterval = "16"
}

Harbour {
 OnTop = "True"
 MaximizeOver = "False"
 Placement = "Right"
 Orientation = "TopToBottom"
 Head = "0"

 DockApp {
  SideMin = "64"
  SideMax = "0"
 }
}
```

Config File Keywords

Here's a table showing the different elements that can be used in your `config` file. Remember that 'boolean' means 'true' or 'false' and that all values should be placed inside quotes.

Config File Elements under the Files-section:

Keys (string)

> The location of the keys file, such as `~/.pekwm/keys`

Menu (string)

> The location of the menu file, such as `~/.pekwm/menu`

Start (string)

> The location of the start file, such as `~/.pekwm/start`

AutoProps (string)

> The location of the autoprops file, such as `~/.pekwm/autoproperties`

Theme (string)

> The location of the Theme directory, such as `~/.pekwm/themes/themename`

Icons (string)

> The location of the Icons directory, such as `~/.pekwm/icons`

Config File Elements under the MoveResize-section:

EdgeAttract (int)

> The distance from screen edge required for the window to snap against it in pixels.

EdgeResist (int)

> The distance from screen edge required for the window moving to start being resisted in pixels.

WindowAttract (int)

> The distance from other clients that a window will snap against them to in pixels.

WindowResist (int)

> The distance from other clients that a window movement will start being resisted.

OpaqueMove (boolean)

> If true, turns on opaque Moving

OpaqueResize (boolean)

> If true, turns on opaque Resizing

Config File Elements under the Screen-section:

Workspaces (int)

> Number of workspaces enabled.

WorkspacesPerRow (int)

> Number of workspaces on each row. Value < 1 fits all workspaces on a single row.

WorkspaceNames (string)

> List of names for workspaces separated by ;.

ShowFrameList (boolean)

> Controls whether a list of all available frames on the workspace is displayed during the
> NextFrame/PrevFrame actions.

ShowStatusWindow (boolean)

> Controls whether a size/position info window is shown when moving or resizing windows.

ShowStatusWindowCenteredOnRoot (boolean)

> Controls whether a size/position info window is shown centered on the current head or the current
> window.

ShowClientID (boolean)

> Should Client IDs be displayed in window titles.

ShowWorkspaceIndicator (int)

> Show WorkspaceIndicator for N milliseconds. If set to < 1, the WorkspaceIndicator is disabled.

WorkspaceIndicatorScale (int)

> Changes the size of the WorkspaceIndicator, higher value means smaller size.

FocusNew (boolean)

> Toggles if new windows should be focused when they pop up.

FocusNewChild (boolean)

> Toggles if new transient windows should be focused when they pop up if the window they are
> transient for is focused.

PlaceNew (boolean)

> Toggles if new windows should be placed using the rules found in the Placement subsection, or just
> opened on the top left corner of your screen.

TrimTitle (string)

> This string contains what pekwm uses to trim down overlong window titles. If it's empty, no
> trimming down is performed at all.

FullscreenAbove (boolean)

Toggles restacking of windows when going to and from fullscreen mode. Windows are restacked to the top of all windows when going to fullscreen and to the top of their layer when being restored from fullscreen.

FullscreenDetect (boolean)

Toggles detection of broken fullscreen requests setting clients to fullscreen mode when requesting to be the size of the screen. Default true.

HonourRandr (boolean)

Toggles reading of XRANDR information, this can be disabled if the display driver gives both Xinerama and Randr information and only of the two is correct. Default true.

HonourAspectRatio (boolean)

Toggles if pekwm respects the aspect ratio of clients (XSizeHints). Default true.

EdgeSize (int) (int) (int) (int)

How many pixels from the edge of the screen should screen edges be. Parameters correspond to the following edges: top bottom left right. A value of 0 disables edges.

EdgeIndent (boolean)

Toggles if the screen edge should be reserved space.

PixmapCacheSize (int)

Determines how many unused pixmaps are stored on the image cache for future use.

DoubleClicktime (int)

Time, in milliseconds, between clicks to be counted as a doubleclick.

Config File Elements under the Placement-subsection of the Screen-section:

Model (string)

- Smart - Tries to place windows where no other window is present
- MouseCentered - Places the center of the window underneath the current mouse pointer position
- MouseTopLeft - Places the top-left corner of the window under the pointer
- MouseNotUnder - Places windows on screen corners avoiding the current mouse cursor position.
- CenteredOnParent - Places transient windows at center of their parent window.

Config File Elements under the Smart-subsection of the Placement-subsection:

Row (boolean)

Whether to use row or column placement, if true, uses row.

TopToBottom (boolean)

If false, the window is placed starting from the bottom.

LeftToRight (boolean)

If false, the window is placed starting from the right.

OffsetX (int)

Pixels to leave between new and old windows and screen edges. When 0, no space is reserved.

OffsetY (int)

Pixels to leave between new and old windows and screen edges. When 0, no space is reserved.

Config File Elements under the UniqueNames-subsection of the Screen-section:

SetUnique (boolean)

Decides if the feature is used or not. False or True.

Pre (string)

String to place before the unique client number.

Post (string)

String to place after the unique client number.

Config File Elements under the CmdDialog-section:

HistoryUnique (boolean)

If true, identical items in the history will only appear once where the most recently used is the first item. Default true.

HistorySize (integer)

Number of entries in the history that should be kept track of. Default 1024.

HistoryFile (string)

Path to history file where history is persisted between session. Default ~/.pekwm/history

HistorySaveInterval (int)

Defines how often the history file should be saved counting each time the CmdDialog finish a command. Default 16.

Config File Elements under the Menu-section:

DisplayIcons (boolean)

> Defines wheter menus should render their icons. Default true.

Icons = "MENU"

> Minimum (width x height)
>
> > Defines minimum size of icons, if 0x0 no check is done. Default 16x16.
>
> Maximum (width x height)
>
> > Defines maximum size of icons, if 0x0 no check is done. Default 16x16.

Select (list of strings)

> Decides on what mouse events to select a menu entry. List is space separated and can include "ButtonPress, ButtonRelease, DoubleClick, Motion, MotionPressed"

Enter (list of strings)

> Decides on what mouse events to enter a submenu. List is space separated and can include "ButtonPress, ButtonRelease, DoubleClick, Motion, MotionPressed"

Exec (list of strings)

> Decides on what mouse events to execute an entry. List is space separated anc can include "ButtonPress, ButtonRelease, DoubleClick, Motion, MotionPressed"

Config File Elements under the Harbour-section:

Placement (string)

> Which edge to place the harbour on. One of Right, Left, Top, or Bottom.

Orientation (string)

> From what to which direction the harbour expands. One of TopToBottom, BottomToTop, RightToLeft, LeftToRight.

OnTop (boolean)

> Whether or not the harbour is "always on top"

MaximizeOver (boolean)

> Controls whether maximized clients will cover the harbour (true), or if they will stop at the edge of the harbour (false).

Head (int)

> When RandR or Xinerama is on, decides on what head the harbour resides on. Integer is the head number.

Config File Elements under the DockApp-subsection of the Harbour-section:

SideMin (int)

> Controls the minimum size of dockapp clients. If a dockapp client is smaller than the minimum, it gets resized up to the SideMin value. Integer is a number of pixels.

SideMax (int)

> Controls the maximum size of dockapp clients. If a dockapp client is larger than the maximum, it gets resized down to the SideMax value. Integer is a number of pixels.

Screen Subsections

There are two subsections in the screen section - Placement and UniqueNames. Placement can optionally have it's own subsection. Sound hard? It's not! It's really quite simple.

We'll start off with Placement. Placement has two options: Model, and a 'Smart' subsection. Model is very simple, it's simply a list of keywords that describes how to place new windows, such as "Smart MouseCentered". Secondly, there's a Smart section, which describes how pekwm computes where to place a new window in smart mode.

The second subsection, UniqueNames, lets you configure how pekwm should handle similar client names. Pekwm can add unique number identifiers to clients that have the same title so that instead of "terminal" and "terminal", you would end up with something like "terminal" and "terminal [2]".

Chapter 9. Configuring the menus

The root menu is what you get when you (by default- See the Mouse Bindings section) left-click on the root window (also called the desktop). You can also configure the window menu, which you get when you right-click on a window title.

Basic Menu Syntax

As previously indicated, the root and window menus follow the rules defined in Common Syntax. There aren't many possible options, and they're all either within the main menu, or within a submenu. This is all handled by a single file.

As addition to the default menu types, RootMenu and WindowMenu, user can define any number of new menus following the same syntax.

Here's an example `~/.pekwm/menu` file, where we have the usual RootMenu and WindowMenu and our own most used applications menu called MyOwnMenuName:

```
# Menu config for pekwm

# Variables
$TERM = "xterm -fn fixed +sb -bg black -fg white"

RootMenu = "Pekwm" {
    Entry = "Term" { Actions = "Exec $TERM &" }
    Entry = "Emacs" { Icon = "emacs.png"; Actions = "Exec $TERM -title emacs -e emacs -nw &
    Entry = "Vim" { Actions = "Exec $TERM -title vim -e vi &" }

    Separator {}

    Submenu = "Utils" {
        Entry = "XCalc" { Actions = "Exec xcalc &" }
        Entry = "XMan" { Actions = "Exec xman &" }
    }

    Separator {}

    Submenu = "Pekwm" {
        Entry = "Reload" { Actions = "Reload" }
        Entry = "Restart" { Actions = "Restart" }
        Entry = "Exit" { Actions = "Exit" }
        Submenu = "Others" {
            Entry = "Xterm" { Actions = "RestartOther xterm" }
            Entry = "Twm" { Actions = "RestartOther twm" }
        }
        Submenu = "Themes" {
            Entry { Actions = "Dynamic ~/.pekwm/scripts/pekwm_themeset.pl" }
        }
    }
}

WindowMenu = "Window Menu" {
```

```
    Entry = "(Un)Stick" { Actions = "Toggle Sticky" }
    Entry = "(Un)Shade" { Actions = "Toggle Shaded" }

    ...

    SubMenu = "Send To" {
        Entry = "Workspace 1" { Actions = "SendToWorkspace 1" }
        Entry = "Workspace 2" { Actions = "SendToWorkspace 2" }
        Entry = "Workspace 3" { Actions = "SendToWorkspace 3" }
        Entry = "Workspace 4" { Actions = "SendToWorkspace 4; GoToWorkspace 4" }
    }

    ...

    Entry = "Close" { Actions = "Close" }
}

MyOwnMenuName = "Most used apps" {
 Entry = "Term" { Actions = "Exec $TERM &" }
 Entry = "XCalc" { Actions = "Exec xcalc &" }
 Entry = "Dillo" { Actions = "Exec dillo &" }
}
```

Menu Keywords

Here are the different elements that can be used in your root menu file.

Root Menu Elements:

Submenu (Name)

> Begins a submenu. 'name' is what will appear in the root menu for the entry.

Entry (Name)

> Begins a menu entry. 'Name' is the text shown in the menu for this entry.

Actions (Action)

> Run an action. 'Action' is the action(s) to run. Most actions listed in Keys/mouse actions will also work from the root and window menus.

Icon (Image)

> Set icon left of entry from image in icon path.

Separator

> Adds a separator to the menu.

Menu Actions:

Exec

> Exec makes pekwm to execute the command that follows it. Make sure the program gets backgrounded. Put an '&' at the end of the action if it doesn't do this on it's own.

Reload

> When this is called, pekwm will re-read all configuration files without exiting.

Restart

> This will cause pekwm to exit and re-start completely.

RestartOther

> Quits pekwm and starts another application. The application to run is given as a parameter.

Exit

> Exits pekwm. Under a normal X setup, This will end your X session.

Of course, in addition to these, many actions found from Keys/mouse actions also work.

Custom Menus

User can also define an unlimited amount of custom menus. They are called with the ShowMenu action much like the Root and Window menus are (see Keys/mouse actions).

In the example menu on this documentation, we created your own menu, called 'MyOwnMenuName'. Basically, outside of the RootMenu and WindowMenu sections, we open our own section called 'MyOwnMenuName'. This can of course be called whatever you want it to be called, but do note that the menu names are case insensitive. This means you can't have one menu called 'MyMostUsedApps' and one called 'mymostusedapps'.

Lets see that example again, simplified:

```
RootMenu = "Pekwm" { ... }
WindowMenu = "Window Menu" { ... }
MyOwnMenuName = "Most used apps" {
 Entry = "Term" { Actions = "Exec $TERM &" }
 Entry = "XCalc" { Actions = "Exec xcalc &" }
 Entry = "Dillo" { Actions = "Exec dillo &" }
}
```

We would call this new menu using the action 'ShowMenu MyOwnMenuName', The menu would show 'Most used apps' as the menu title and list 'Term', 'XCalc' and 'Dillo' in the menu ready to be executed.

Dynamic Menus

It is possible to use dynamic menus in pekwm, that is menus that regenerate themselves whenever the menu is viewed. This is done with the Dynamic keyword.

To use this feature, you need to put a dynamic entry in the `~/.pekwm/menu` file with a parameter that tells pekwm what file to execute to get the menu. This file can be of any language you prefer, the main thing is that it outputs valid pekwm menu syntax inside a Dynamic {} section. The syntax of dynamic entry looks like this:

```
Entry = "" { Actions = "Dynamic /path/to/filename" }
```

The input from a program that creates the dynamic content should follow the general menu syntax encapsulated inside a Dynamic {} section. Variables have to be included inside the dynamic menu for them to work. A simple script to give pekwm dynamic menu content would look like this:

```
#!/bin/bash
output=$RANDOM # gets a random number

echo "Dynamic {"
echo " Entry = \"$output\" { Actions = \"Exec xmessage $output\" }"
echo "}"
```

This script would output something like:

```
Dynamic {
 Entry = "31549" { Actions = "Exec xmessage 31549" }
}
```

Clients can access the PID and Window that was active when the script was executed via the environment variables $CLIENT_PID and $CLIENT_WINDOW. $CLIENT_PID is only available if the client is being run on the same host as pekwm.

Chapter 10. Autoproperties

What are Autoproperties?

"Autoproperties" is short for "Automatic Properties". This is pekwm's way of setting certain things up for applications based on the window's internal id. You can set up a lot of things, such as size, iconified state, start location, grouped state (automatically having one window group to another), workspace to start on, whether it has a border or titlebar, and more. It is also possible to automatically modify window titles and to decide the order of applications on the harbour with autoproperties.

Basic Autoproperties Syntax

The `~/.pekwm/autoproperties` file follows the rules in Common Syntax. This file can become rather complicated, but it's also the most powerful of any of pekwm's config files.

The one important thing to remember is the Property tag. This identifier tells us where to apply properties. It means which windows to apply it on. To find out the two terms, use **xprop WM_CLASS** and click on your window. Below you'll find a bash/zsh function which will give you the correct string for this file. You can also specify a regexp wildcard, such as ".*,opera", which means anything for the first word, opera for the second.

```
propstring () {
  echo -n 'Property '
  xprop WM_CLASS | sed 's/.*"\(.*\)", "\(.*\)".*/= "\1,\2" {/g'
  echo '}'
}
```

Autoproperties have an both an old and new style matching clients. The new style was introduced to support using configuration template overwriting.

In addition with WM_CLASS, pekwm also can identify clients by their title string (**xprop WM_NAME** or **xprop _NET_WM_NAME**).

```
# New syntax, requires Require { Templates = "True" }
Property = "^dillo,^Dillo„Dillo: pekwm.org - not just another windowmanager" {
 ApplyOn = "Start New"
 Layer = "OnTop"
}

# Old syntax
Property = "^dillo,^Dillo" {
        Title = "Dillo: pekwm.org - not just another windowmanager"
 ApplyOn = "Start New"
 Layer = "OnTop"
}
```

Or by their role (**xprop WM_WINDOW_ROLE**):

```
# New syntax, requires Require { Templates = "True" }
Property = "^gaim,^Gaim,preferences" {
```

```
 ApplyOn = "New"
 Skip = "Menus"
}

# Old syntax
Property = "^gaim,^Gaim" {
 Role = "preferences"
 ApplyOn = "New"
 Skip = "Menus"
}
```

Pekwm can rewrite window titles. This is done in a separate TitleRules section, where one defines properties on which clients to use the rewriting and then a regexp rule of what to do to that clients title. These rules do not affect the actual WM_NAME string. You can use Role and Title keywords to narrow down the clients the titlerule applies to. A simple rule that would change "Title: this is the title" to "this is the title" looks like this:

```
TitleRules {
 Property = "^foo,^bar" {
  Rule = "/Title: (.*)/\\1/"
 }
}
```

In pekwm, you can make certain windows have their own decoration set. The different decorations are defined in the theme, and they are connected to client windows with an autoproperty. These autoproperties reside in their own DecorRules subsection and look like this:

```
DecorRules {
 Property = "^foo,^bar" {
  Decor = "TERM"
 }
}
```

It's also possible to decide the order of applications that start in the harbour. As with TitleRules and DecorRules, there is it's own separate section for this purpose called Harbour. Position is a signed int and order goes: "1 2 3 0 0 0 -3 -2 -1", and so on. That looked cryptic. Worry not. Basically, a Position number of 0 means the application will be placed in the middle. If the number is positive, the application will be placed before the zero-positioned applications. If the number is negative, they applications will be placed after the zero-position ones. So the positive numbered show up first in your harbour, then the zero numbered, and after the zeros come the negatively numbered applications. I hope that is clear, the next part is tricky. The larger the value of the base number the closer to the zero applications they will be. So the smaller the base number the closer to the ends of the harbour the application will be. Position 1 would be the first application to show up on the harbour. And similarly Position -1 would be the last application on the harbour. If you have application on the harbour that do not match any of the property rules on the Harbour section, they will act as if you had given them Position 0. Applications with the same Position will show up next to each other in the order they are launched. In our example below, obpager will always be placed the last on the harbour.

```
Harbour {
 Property = "^obpager,^obpager" {
  Position = "-1";
```

```
 }
}
```

Here's an example `~/.pekwm/autoproperties` file:

```
Property = ".*,^xmms" {
 ApplyOn = "Start New"
 Layer = "0"
 Sticky = "True"
}

Property = "^xclock,^XClock" {
 ApplyOn = "Start New"
 FrameGeometry = "100x100+0-0"
 Border = "False"; Titlebar = "False"
 Sticky = "True"
 Layer = "Desktop"
}

Property = "^dillo,^Dillo" {
 ApplyOn = "Start New"
 Group = "browsers" {
  Size = "30"
  Behind = "True"
  Global = "False"
 }
}

TitleRules {
 Property = "^dillo,^Dillo" {
  Rule = "/Dillo: (.*)/\\1 [dillo]/"
 }
 Property = "^opera,^opera" {
  Rule = "/...:... - (.*) - Opera .*/\\1 [opera]/"
 }
}

DecorRules {
 Property = "^.term,^XTerm" {
  Decor = "TERM"
 }
}

Harbour {
 Property = "^obpager,^obpager" {
  Position = "-1"
 }
}
```

Regular Expressions! The pekwm autoproperties file uses Regular Expression syntax for wildcards. Regular expressions can be really confusing to people with no experience with them. A good rule of thumb is: "Anywhere you'd think to use '*', use '.*'". Also, '^' matches the beginning of a

string, '$' matches the end, and '.' is any single character. Pekwm has some special flags to that modifies regular expression matching. Specifying regular expressions in the form /pattern/flags allow flags to be set. The supported flags are ! for inverting the match and i for case insensitive matches. Regular Expression syntax is a little bit beyond the scope of this documentation. There's a really basic tutorial at http://www.living-source.com/user/adi/regexp/regexp_tutorial.html, And a slightly more advanced one at http://www.zytrax.com/tech/web/regex.htm. You might also want to look at http://www.regularexpressions.info/ I found these on google, using the search phrase "regular.expression.tutorial". You can probably use that search term or modify it to find some others.

Advanced Autoproperties

Below is a list of the different actions available to you in your autoproperties file; These are the actual Auto Properties. They can take four types of arguments: bool, integer, string, or geom. A bool is either True (1) or False (0). An Integer is a number, negative or positive. A string is any string, it's used as an identifier. Finally, geom is an X Geometry String by the form:
"[=][<width>{xX}<height>][{+-}<xoffset>{+-}<yoffset>]" (see: man 3 XParseGeometry). Examples are 200x300+0+0, 0x500+200+300, 20x10+0+50, et cetera.

Exhaustive Autoprops List

Sticky (bool)

   Window starts Sticky (present on all workspaces)

Shaded (bool)

   Window starts Shaded

MaximizedVertical (bool)

   Window starts Maximized Vertically

MaximizedHorizontal (bool)

   Window starts Maximized Horizontally

Iconified (bool)

   Window starts Iconified

Border (bool)

   Window starts with a border

Titlebar (bool)

   Window starts with a TitleBar

FrameGeometry (geom)

   X Geometry String showing the initial size and position of the window frame. Window frame
   includes the client window and the possible pekwm titlebar and window borders. If both

ClientGeometry and FrameGeometry are present, FrameGeometry overrides the ClientGeometry.

ClientGeometry (geom)

X Geometry String showing the initial size and position of the client, excluding the possible pekwm titlebar and window borders.

Layer (string)

Windows layer. Makes the window stay under or above other windows. Default layer is "Normal". Possible parameters are (listed from the bottommost to the uppermost):

- Desktop
- Below
- Normal
- OnTop
- Harbour
- AboveHarbour
- Menu

Workspace (integer)

Which workspace to start program on.

Skip (string)

A list of situations when to ignore the defined application and let the user action skip over it, consisting of

- "Snap" (Do not snap to this window while moving windows)
- "Menus" (Do not show this window in pekwm menus other than the icon menu)
- "FocusToggle" (Do not focus to this window when doing Next/PrevFrame)

Fullscreen (bool)

Window starts in fullscreen mode

PlaceNew (bool)

Toggles the use of placing rules for this client.

FocusNew (bool)

Toggles if this client gets focused when it initially pops up a window.

Focusable (bool)

Toggles if this client can be focused while it's running.

CfgDeny (string)

A list of conditions of when to deny things requested by the client program, consisting of

- "Position" (Ignore client requested changes to window position)

- "Size" (Ignore client requested changes to window size)

- "Stacking" (Ignore client requested changes to window stacking)

- "ActiveWindow" (Ignore client requests for showing and giving input focus)

- "MaximizedVert" (Ignore client request to maximize window vertically)

- "MaximizedHorz" (Ignore client request to maximize window horizontally)

- "Hidden" (Ignore client request to show/hide window)

- "Fullscreen" (Ignore client request to set window to fullscreen mode)

- "Above" (Ignore client request to always place window above other windows)

- "Below" (Ignore client request to always place window below other windows)

ApplyOn (string)

A list of conditions of when to apply this autoprop (so be sure to include this in your property), consisting of

- "Start" (Apply if window already exists before pekwm starts/restarts. Note when using grouping Start will not take workspaces in account)

- "New" (Applies when the application first starts)

- "Reload" (Apply when pekwm's config files are reloaded)

- "Workspace" (Apply when the window is sent to another workspace)

- "Transient" (Apply to Transient windows as well as normal windows. Dialog boxes are commonly transient windows)

- "TransientOnly" (Apply to Transient windows only. Dialog boxes are commonly transient windows)

Title (string)

Apply this autoproperty on clients that have a title that matches this string. String is a regexp like: "^Saving".

Role (string)

Apply this autoproperty on clients that have a WM_WINDOW_ROLE hint that matches this string. String is a regexp like: "^Main".

Group (string)

Defines the name of the group. Also the section that contains all the grouping options. They are:

- Size (integer) - How many clients should be grouped in one group.

- Behind (bool) - If true makes new clients of a group not to become the active one in the group.

- FocusedFirst (bool) - If true and there are more than one frame where the window could be autogrouped into, the currently focused frame is considered the first option.

- Raise (bool) - If true makes new clients raise the frame they open in.

- Global (bool) - If true makes new clients start in a group even if the group is on another workspace or iconified.

AutoGrouping

The last thing to know is autogrouping. Autogrouping is actually very simple, although it might be a bit confusing at first. Group is an identifier, it's just a string, (in my example, we'll call it netwin). Size tells how many clients to group together in one frame.

The example: We want to autogroup Sylpheed and Opera together, allowing as many instances of the program windows to be grouped as there are. Here's the Autoprops section for that:

```
Property = ".*,^opera" {
 Group = "netwin" {
  Size = "0"
 }
 ApplyOn = "New Start Reload"
}
Property = ".*,^Sylpheed" {
 Group = "netwin" {
  Size = "0"
 }
 ApplyOn = "New Start Reload Transient"
}
```

This creates two rules: "For any window matching '.*,^opera', group these in the 'netwin' group. Apply this on pekwm start/reload and when new windows matching this property are opened, but do not include dialog windows", and "For any window matching '.*,^Sylpheed', group in the 'netwin' group. Apply on pekwm start/reload and when new windows matching this property are opened, also include possible dialog windows to the group. Open the window to the group but do not bring it upmost automatically".

To group unlimited windows together, use size 0.

Also note that you can have as many Group identifiers as you want. Autogrouping is a very flexible system. Try playing around with it.

TypeRules, autoproperties controlling _NET_WM_WINDOW_TYPE

The TypeRules decides how the _NET_WM_WINDOW_TYPE should be interpreted. The _NET_WM_WINDOW_TYPE hint gives the application writer possibility to inform the window manager what kind of window it is creating.

TypeRules are defined in the TypeRules section of the `~/.pekwm/autoproperties` file. A sample section could look something like this:

```
TypeRules {
    ...

    Property = "MENU"  {
        Titlebar = "False"
        Border = "False"
        Skip = "FocusToggle Menus Snap"
    }

    ...
}
```

Using TypeRules are done the same way as with Advanced Autoproperrties but the property is matched based on the value of _NET_WM_WINDOW_TYPE. Supported values are available in the list below.

Supported values

Desktop

    A desktop window such as the window containing desktop icons on the Gnome desktop.

Dock

Toolbar

Menu

Utility

Splash

    Application startup screen usually presenting loading progress.

Dialog

    Dialogs prompting for information such as "Save as" dialogs.

Normal

    Any other window, can be used to set default autoproperties.

Getting more help

Autoprops can be a daunting topic. If you've read everything here in the docs and are still having problems, feel free to hit the IRC channel and ask. Check the Common questions and answers before

asking. Remember that: "IF YOU WANT AN ANSWER TO YOUR QUESTION, YOU HAD BETTER HAVE ALREADY READ THE DOCUMENTATION".

Chapter 11. Keyboard and Mouse Configuration

Pekwm allows you to remap almost all keyboard and mouse events.

Mouse Bindings

The pekwm Mousebindings go in `~/.pekwm/mouse`, and are very simple. They're divided up into two groups: The 'where' and 'event'. Below is an example file:

```
FrameTitle {
 ButtonRelease = "1" { Actions = "Raise; Focus; ActivateClient" }
 ButtonRelease = "2" { Actions = "ActivateClient" }
 ButtonRelease = "Mod4 3" { Actions = "Close" }
 ButtonRelease = "3" { Actions = "ShowMenu Window" }
 ButtonRelease = "4" { Actions = "ActivateClientRel 1" }
 ButtonRelease = "5" { Actions = "ActivateClientRel -1" }
 DoubleClick = "2" { Actions = "Toggle Shaded" }
 DoubleClick = "1" { Actions = "MaxFill True True" }
 Motion = "1" { Threshold = "4"; Actions = "Move" }
 Motion = "Mod1 1" { Threshold = "4"; Actions = "Move" }
 Motion = "Mod4 1" { Threshold = "4"; Actions = "Move" }
 Motion = "2" { Threshold = "4"; Actions = "GroupingDrag True" }
 Motion = "Mod1 3" { Actions = "Resize" }
 Enter = "Any Any" { Actions = "Focus" }
}

OtherTitle {
 ButtonRelease = "1" { Actions = "Raise; Focus; ActivateClient" }
 ButtonRelease = "Mod4 3" { Actions = "Close" }
 DoubleClick = "2" { Actions = "Toggle Shaded" }
 DoubleClick = "1" { Actions = "MaxFill True True" }
 Motion = "1" { Threshold = "4"; Actions = "Move" }
 Motion = "Mod1 1" { Threshold = "4"; Actions = "Move" }
 Motion = "Mod4 1" { Threshold = "4"; Actions = "Move" }
 Motion = "Mod1 3" { Actions = "Resize" }
 Enter = "Any Any" { Actions = "Focus" }
}

Border {
 TopLeft     { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Resi
 Top         { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Move
 TopRight    { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Resi
 Left        { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Resi
 Right       { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Resi
 BottomLeft  { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Resi
 Bottom      { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Resi
 BottomRight { Enter = "Any Any" { Actions = "Focus" }; ButtonPress = "1" { Actions = "Resi
}

ScreenEdge {
 Down {
```

```
  ButtonRelease = "3" { Actions = "ShowMenu Root" }
  ButtonRelease = "2" { Actions = "ShowMenu Goto" }
 }
 Up {
  ButtonRelease = "3" { Actions = "ShowMenu Root" }
  ButtonRelease = "2" { Actions = "ShowMenu Goto" }
  ButtonRelease = "Mod1 4" { Actions = "GoToWorkspace Right" }
  ButtonRelease = "Mod1 5" { Actions = "GoToWorkspace Left" }
 }
 Left {
  Enter = "Mod1 Any" { Actions = "GoToWorkspace Left" }
  ButtonRelease = "3" { Actions = "ShowMenu Root" }
  ButtonRelease = "1" { Actions = "GoToWorkspace Left" }
  DoubleClick = "1" { Actions = "GoToWorkspace Left" }
  ButtonRelease = "2" { Actions = "ShowMenu Goto" }
  ButtonRelease = "4" { Actions = "GoToWorkspace Right" }
  ButtonRelease = "5" { Actions = "GoToWorkspace Left" }
 }
 Right {
  Enter = "Mod1 Any" { Actions = "GoToWorkspace Right" }
  ButtonRelease = "3" { Actions = "ShowMenu Root" }
  ButtonRelease = "1" { Actions = "GoToWorkspace Right" }
  DoubleClick = "1" { Actions = "GoToWorkspace Right" }
  ButtonRelease = "2" { Actions = "ShowMenu Goto" }
  ButtonRelease = "4" { Actions = "GoToWorkspace Right" }
  ButtonRelease = "5" { Actions = "GoToWorkspace Left" }
 }
}

Client {
 ButtonPress = "1" { Actions = "Focus" }
 ButtonRelease = "Mod1 1" { Actions = "Focus; Raise" }
 ButtonRelease = "Mod4 1" { Actions = "Lower" }
 Motion = "Mod1 1" { Actions = "Focus; Raise; Move" }
 Motion = "Mod4 1" { Actions = "Focus; Raise; Move" }
 Motion = "Mod1 2" { Threshold = "4"; Actions = "GroupingDrag True" }
 Motion = "Mod1 3" { Actions = "Resize" }
 Enter = "Any Any" { Actions = "Focus" }
}

Root {
 ButtonRelease = "3" { Actions = "ShowMenu Root" }
 ButtonRelease = "2" { Actions = "ShowMenu Goto" }
 ButtonRelease = "4" { Actions = "GoToWorkspace Right" }
 ButtonRelease = "5" { Actions = "GoToWorkspace Left" }
 ButtonRelease = "1" { Actions = "HideAllMenus" }
}

Menu {
 Enter = "Any Any" { Actions = "Focus" }
 ButtonRelease = "2" { Actions = "Toggle Sticky" }
 Motion = "1" { Threshold = "10"; Actions = "Move" }
 ButtonRelease = "3" { Actions = "Close" }
```

```
}

Other {
 Enter = "Any Any" { Actions = "Focus" }
 ButtonRelease = "Mod4 3" { Actions = "Close" }
 Motion = "1" { Actions = "Focus; Raise; Move" }
 Motion = "Mod1 1" { Actions = "Focus; Raise; Move" }
}
```

Below are defined the different fields. The actions themselves can be found in the Keys/mouse actions section.

'Where' fields:

FrameTitle

> On a regular window's Titlebar.

OtherTitle

> On menu/cmdDialog/etc pekwm's own window's Titlebar.

Border

> On the window's borders. See Border Subsection for more information.

ScreenEdge

> On the screen edges. See ScreenEdge Subsection for more information.

Client

> Anywhere on the window's interior. It's best to use a keyboard modifier with these.

Root

> On the Root window (also called the 'desktop').

Menu

> On the various menus excluding their titlebars.

Other

> On everything else that doesn't have it's own section.

'Event' fields:

ButtonPress

> A single click

ButtonRelease

> A single click that activates once the button is released

DoubleClick

> A double click

Motion

> Clicking, holding, and Dragging.

Enter

> Defines how to act when mouse pointer enters a place defined by the 'where' field.

Leave

> Defines how to act when mouse pointer leaves a place defined by the 'where' field.

EnterMoving

> Defines how to act when a dragged window enters a ScreenEdge. Only works with screen edges.

Definitions work like this:

```
'Where' {
 'Event' = "optional modifiers, like mod1, ctrl, etc and a mouse button" {
  Actions = "actions and their parameters"
 }
 'Event' = "optional modifiers, like mod1, ctrl, etc and a mouse button" {
  Actions = "actions and their parameters"
 }
}
```

> **Additional notes.** Modifiers and mouse buttons can be defined as "Any" which is useful for Enter
> and Leave events. Any also applies as none. Motion events have a threshold argument. This is the
> number of pixels you must drag your mouse before they begin to work. Multiple actions can be
> defined for a single user action. Example:

```
Motion = "1" { Actions = "Move"; Treshold = "3" }
ButtonPress = "1" { Actions = "Raise; ActivateClient" }
```

Border Subsection

The Border subsection in `~/.pekwm/mouse` defines the actions to take when handling the window
borders.

```
Border {
 TopLeft {
  Enter = "Any Any" { Actions = "Focus" }
```

```
  ButtonPress = "1" { Actions = "Resize TopLeft" }
 }
}
```

It's subsections refer to the frame part in question. They are: Top, Bottom, Left, Right, TopLeft, TopRight, BottomLeft, and BottomRight. In these subsections you can define events and actions as usual.

ScreenEdge Subsection

The ScreenEdge subsection in `~/.pekwm/mouse` defines the actions to take when an event happens on the specified screenedge.

```
ScreenEdge {
    Left {
        Enter = "Mod1 Any" { Actions = "GoToWorkspace Left" }
        ButtonPress = "3" { Actions = "ShowMenu Root" }
        ButtonPress = "1" { Actions = "GoToWorkspace Left" }
        ButtonPress = "2" { Actions = "ShowMenu Goto" }
        ButtonPress = "4" { Actions = "GoToWorkspace Right" }
        ButtonPress = "5" { Actions = "GoToWorkspace Left" }
    }
}
```

It has four subsections: Up, Down, Left, and Right, that all refer to the screen edge in question. In these subsections you can give events and actions as usual.

Key Bindings

The pekwm keybindings go in `~/.pekwm/keys`, and are even more simple than the mouse bindings. Here's the format:

```
KeyPress = "optional modifiers like mod1, ctrl, etc and the key" {
 Actions = "action and the parameters for the action, if they are needed"
}
```

Multiple actions can be given for one keypress. The actions are separated from each other with a semicolon:

```
Keypress = "Ctrl t" { Actions = "Exec xterm; Set Maximized True True; Close" }
```

Here's a small fragment of an example keys file; you can see a full version in `~/.pekwm/keys`. As with the mouse, you can see the full list of actions in the Keys/mouse actions section.

```
Global {
 # Moving in frames
 KeyPress = "Mod1 Tab" { Actions = "NextFrame EndRaise" }
 KeyPress = "Mod1 Shift Tab" { Actions = "PrevFrame EndRaise" }
 KeyPress = "Mod1 Ctrl Tab" { Actions = "NextFrameMRU EndRaise" }
 KeyPress = "Mod1 Ctrl Shift Tab" { Actions = "PrevFrameMRU EndRaise" }
 # Simple window management
 KeyPress = "Mod4 M" { Actions = "Toggle Maximized True True" }
```

```
 KeyPress = "Mod4 G" { Actions = "Maxfill True True" }
 KeyPress = "Mod4 F" { Actions = "Toggle FullScreen" }
 KeyPress = "Mod4 Return" { Actions = "MoveResize" }
 # Wm actions
 Chain = "Ctrl Mod1 P" {
  KeyPress = "Delete" { Actions = "Reload" }
  KeyPress = "Next" { Actions = "Restart" }
  KeyPress = "End" { Actions = "Exit" }
  KeyPress = "Prior" { Actions = "RestartOther twm" }
 }
}

MoveResize {
 KeyPress = "Left" { Actions = "MoveHorizontal -10" }
 KeyPress = "Right" { Actions = "MoveHorizontal 10" }
 KeyPress = "Up" { Actions = "MoveVertical -10" }
 KeyPress = "Down" { Actions = "MoveVertical 10" }
 Keypress = "Mod4 Left" { Actions = "ResizeHorizontal -10" }
 Keypress = "Mod4 Right" { Actions = "ResizeHorizontal 10" }
 Keypress = "Mod4 Up" { Actions = "ResizeVertical -10" }
 Keypress = "Mod4 Down" { Actions = "ResizeVertical 10" }
 Keypress = "s" { Actions = "MoveSnap" }
 Keypress = "Escape" { Actions = "Cancel" }
 Keypress = "Return" { Actions = "End" }
}

Menu {
 KeyPress = "Down" { Actions = "NextItem" }
 KeyPress = "Up" { Actions = "PrevItem" }
 KeyPress = "Left" { Actions = "LeaveSubmenu" }
 KeyPress = "Right" { Actions = "EnterSubmenu" }
 KeyPress = "Return" { Actions = "Select" }
 KeyPress = "Escape" { Actions = "Close" }
}

InputDialog {
 KeyPress = "BackSpace" { Actions = "Erase" }
 KeyPress = "Right" { Actions = "CursNext" }
 KeyPress = "Left" { Actions = "CursPrev" }
 KeyPress = "Up" { Actions = "HistPrev" }
 KeyPress = "Down" { Actions = "HistNext" }
 KeyPress = "Delete" { Actions = "Clear" }
 KeyPress = "Return" { Actions = "Exec" }
 KeyPress = "Escape" { Actions = "Close" }
 KeyPress = "Any Any" { Actions = "Insert" }
}
```

As you might have noticed, the file consist of four sections. These sections are Global, MoveResize, Menu and InputDialog. The first section, Global, contains all the generic actions.

The MoveResize section has the keybindings that will get used when the MoveResize action is called.

Menu section contains the keys that are used when the ShowMenu action is called. E.g. these are the keys you use to browse thru the menus of pekwm. Note that while ShowMenu is active, the Global

keybindings are also listened. If a keybinding is same in both Menu and Global sections, keybindings in Menu section override the global keybinding as long as a menu is active.

Finally, the InputDialog section allow for tuning of what keys are available for line editing when the CmdDialog window that enables the user to enter pekwm actions for running windows is active.

Keys can be identified with their XString name or with their keycode. Both can be found out using the X application **xev**. If you want to use a keycode, prefix it with **#**.

Keychains

Pekwm also supports keychains. Keychain syntax follows the general config syntax and looks like this:

```
Chain = "modifiers and key" {
 Chain = "modifiers and key" {
  KeyPress = "modifiers and key" { Actions = "actions and their parameters" }
 }
 Keypress = "modifiers and key" { Actions = "actions and their parameters" }
}
```

It might seem complicated at start but once you look into it, it is fairly nice and logical. This syntax supports as many nested Chains as you might want.

Now for some examples. Here we have a simple nested chain that lets you press Ctrl+Alt+M, then M, then M, V or H to toggle maximized attribute into Full/Vertical or Horizontal, and a simpler one level chain that brings up the root menu.

```
Chain = "Ctrl Mod1 A" {
 Chain = "M" {
  KeyPress = "M" { Actions = "Toggle Maximized True True" }
  KeyPress = "V" { Actions = "Toggle Maximized False True" }
  KeyPress = "H" { Actions = "Toggle Maximized True False" }
 }
}
Chain = "Ctrl Mod1 M" {
 KeyPress = "R" { Actions = "ShowMenu Root" }
}
```

This next rule is a pure show-off, it lets you type in 'test' and then executes xterm. Note that this will make you unable to type the character 't' to any programs.

```
Chain = "t" { Chain = "e" { Chain = "s" {
 Keypress = "t" { Actions = "Exec xterm" }
} } }
```

Keys/Mouse actions and window attributes

Here is the list of all possible actions and attributes. First table shows all toggleable attributes. Toggleable attributes are controlled using the *Set*, *Unset* and *Toggle* actions. Examples below.

```
 Keypress = "Mod4 s"       { Actions = "Toggle Shaded" }
 Keypress = "Mod4 m"       { Actions = "Toggle Maximized True True" }
```

```
Keypress = "Mod4 t"       { Actions = "Set Tagged" }
Keypress = "Mod4 Shift t" { Actions = "Unset Tagged" }
```

Toggleable attributes:

Maximized (bool bool)

> If a frame is maximized. Two parameters, first one decides if the frame is maximized horizontally, the second if it is maximized vertically.

Fullscreen

> If a frame should be fullscreen. Fullscreen frame takes over the whole desktop ignoring any harbour or struts and becomes decorless.

Shaded

> If a frame should be shaded (so that only the titlebar shows until it's unset or toggled off).

Sticky

> If a frame should be sticky so it appears on every workspace.

AlwaysOnTop

> If frame should always be on top of other frames.

AlwaysBelow

> If a frame should always be below other frames.

DecorBorder

> If frame should have borders.

DecorTitlebar

> If frame should have a titlebar.

Iconified

> If a frame should be iconified.

Tagged (bool)

> If a frame should swallow all new clients until unset or toggled off. One parameter, if true new clients open in the background. Defaults to false.

Marked

> If a frame is marked for later attaching (with AttachMarked).

Skip (string)

> If a frame should be ignored on specified places, string is one of

> - menus

- focustoggle

- snap

CfgDeny (string)

When things to be done to this window requested by the client program should be denied, string is one of:

- position (don't let the client move the window)

- size (don't let the client resize the window)

- stacking (don't allow the client to raise or lower the window)

- activewindow (don't let client give input focus)

- maximizedvert (don't let client maximize a window vertically)

- maximizedhorz (don't let client maximize window horizontally)

- hidden (don't let client hide window)

- fullscreen (don't let client set window fullscreen mode)

- above (don't let client place window above other windows)

- below (don't let client place window below other windows)

Title (string)

Changes the clients titlebar text to string when set. Unsetting returns the clients title text back to the client specified one.

HarbourHidden

If set, harbour and anything in it will be hidden from the screen.

GlobalGrouping

If all autogrouping should be in use or not. By default it's set, as in autogrouping is enabled.

Keys/Mouse Actions:

Focus

Gives focus to a frame.

UnFocus

Removes focus from a frame.

Set (one of toggleable attributes)

Makes toggleable attributes set.

UnSet (one of toggleable attributes)

> Unsets toggleable attributes.

Toggle (one of toggleable attributes)

> Toggles toggleable attributes.

MaxFill (bool bool)

> Acts much like Maximize, but considers other frames while doing it. Instead of filling the whole screen, it only fills to the borders of neighboring frames. Takes two parameters, first one decides if the frame is maxfilled horizontally, the second if it should be maxfilled vertically.

GrowDirection (string)

> Grows the frame in one of the directions up to the edge of the head. String is one of up, down, left, right.

Close

> Closes a client window.

CloseFrame

> Closes a frame and all client windows in it.

Kill

> Kills a client window, use if close doesn't work.

Raise (bool)

> Raises a frame above other frames. If bool is true raises a frame and all of the currently active clients child/parent windows above other frames.

Lower (bool)

> Lowers a frame under other frames. If bool is true lowers a frame and all of the currently active clients child/parent windows under other frames.

ActivateOrRaise

> If the frame this action is used on is not focused, focuses it. If the frame is focused, raises it. If used on a groups titlebar, activates the selected client of the group.

ActivateClientRel (int)

> Moves the focus and raises a client inside a frame relative to the currently selected client. Int is 1 to move right, -1 to move left.

MoveClientRel (int)

> Moves the current clients position inside the current frame. Int is 1 to move right, -1 to move left.

ActivateClient

> Activates a client of a frame.

Mouse-specific

ActivateClientNum (int)

Activates the #th client of a frame. Int is the client number counting from left.

Keygrabber-specific

Resize (string)

Resizes a frame. String is one of top, bottom, left, right, topleft, topright, bottomleft, bottomright.

Mouse-specific (parameters frameborder-specific)

Move

Moves a frame.

Mouse-specific

MoveResize

Activates the keyboard move and resize.

Keygrabber-specific

GroupingDrag (bool)

Drags windows in and out of frames, if parameter is true dragged windows go in the background of a frame.

Mouse-specific

WarpToWorkspace (string)

Makes a dragged window warp to specified workspace when a it's moved over a screen edge. String is one:

- next - send to the next workspace, if on last workspace, wrap to the first one.

- prev - send to the previous workspace, if on first workspace, wrap to the last one.

- left - send to the previous workspace

- right - send to the next workspace

- int - integer is a workspace number to send to to

ScreenEdge specific mouse binding

MoveToEdge (string)

Moves the frame to the specified place on the screen. String is one of TopLeft, TopEdge, TopRight, RightEdge, BottomRight, BottomEdge, BottomLeft, LeftEdge, Center, TopCenterEdge, BottomCenterEdge, LeftCenterEdge, RightCenterEdge.

Keygrabber-specific

NextFrame (string boolean)

Focuses the next frame. String is one of:

- alwaysraise - raise windows while toggling them

- endraise - raise the selected client

- neverraise - do not raise the selected client (unless it's iconified)

If boolean is true, also goes thru iconified windows. Defaults to false.

PrevFrame (string boolean)

Focuses the previous frame. String is:

- alwaysraise - raise windows while toggling them

- endraise - raise the selected client

- neverraise - do not raise the selected client (unless it's iconified)

If boolean is true, also goes thru iconified windows. Defaults to false.

NextFrameMRU (string boolean)

Focuses the next frame so that the last focused windows will get switched to first. String is:

- alwaysraise - raise windows while toggling them

- endraise - raise the selected client

- neverraise - do not raise the selected client (unless it's iconified)

If boolean is true, also goes thru iconified windows. Defaults to false.

PrevFrameMRU (string boolean)

Focuses the previous frame so that the last focused windows will get switched to first. String is:

- alwaysraise - raise windows while toggling them

- endraise - raise the selected client

- neverraise - do not raise the selected client (unless it's iconified)

If boolean is true, also goes thru iconified windows. Defaults to false.

FocusDirectional (string bool)

Focuses the first window on the direction specified, and optionally raises it. Takes two options, first one is the direction and the second specifies if the focused frame should be raised or not. Bool defaults to True. String is one of up, down, left, right

AttachMarked

Attachs all marked clients to the current frame.

AttachClientInNextFrame

Attachs client to the next frame.

AttachClientInPrevFrame

> Attachs client to the previous frame.

FindClient (string)

> Searches the client list for a client that has a title matching the given regex string.

GotoClientID (string)

> Shows and focuses a client based on the Client ID given as a parameter.

Detach

> Detach the current client from its frame.

SendToWorkspace (string)

> Sends a frame to the specified workspace. String is one of:
>
> - next - send to the next workspace, if on last workspace, wrap to the first one.
> - prev - send to the previous workspace, if on first workspace, wrap to the last one.
> - left - send to the previous workspace
> - right - send to the next workspace
> - prevv - send to the previous (vertical) workspace, if on last workspace, wrap to the first one.
> - up - send to the previous (vertical) workspace.
> - nextv - sed to the next (vertical) workspace, if on last workspace, wrap to the first one.
> - down -
> - last - send to workspace you last used before the current
> - int - integer is a workspace number to send to to

GotoWorkspace (string)

> Changes workspaces. String is one of:
>
> - left - go to the previous workspace
> - prev - go to the previous workspace, if on first workspace, wrap to the last one.
> - right - go to the next workspace
> - next - go to the next workspace, if on last workspace, wrap to the first one.
> - prevv -
> - up -
> - nextv -
> - down -
> - last - go to workspace you last used before the current
> - int - integer is a workspace number to go to

Exec (string)

> Executes a program, string is a path to an executable file.

Reload

> Reloads pekwm configs.

> Keygrabber-specific

Restart

> Restarts pekwm.

> Keygrabber-specific

RestartOther

> Quits pekwm and starts the program you specify. String is a path to an executable file.

> Keygrabber-specific

Exit

> Exits pekwm.

ShowCmdDialog (string)

> Shows the command dialog that can be used to input pekwm actions. If it's a window specific action, it affects the window focused when CmdDialog was summoned. If entered action doesn't match any valid pekwm action, pekwm tries to Exec it as a shell command. Takes an optional string as a parameter. This string will then be pre-filled as the initial value of the dialog.

ShowSearchDialog (string)

> Shows the search dialog that can be used to search for clients and when selected the client will be activated. Takes an optional string as a parameter. This string will then be pre-filled as the initial value of the dialog.

ShowMenu (string bool)

> Shows a menu. String is menu type from below list or user defined menu name (see Custom Menus):

> - root - shows your application menu
> - icon - shows iconified windows
> - goto - shows currently active clients
> - gotoclient - shows all open clients
> - window - shows a window specific menu
> - decor - shows possible decorations in the current theme
> - attachclient - allows to attach clients in current frame
> - attachframe - allows to attach whole frame in current frame
> - attachclientinframe - allows attaching current client in any other frame

- attachframeinframe - allows attaching current frame in any other frame

   Bool is true for sticky menus, false for click to vanish. Defaults to false.

HideAllMenus

   Closes all pekwm menus.

SendKey

   Send a key, possibly with modifiers, to the active window.

MoveResize actions:

MoveHorizontal (int)

   Moves a frame horizontally. Int is amount of pixels and can be negative.

   Moveresize-specific keybinding

MoveVertical (int)

   Moves a frame vertically. Int is amount of pixels and can be negative.

   Moveresize-specific keybinding

ResizeHorizontal (int)

   Resizes a frame horizontally. Int is amount of pixels and can be negative.

   Moveresize-specific keybinding

ResizeVertical (int)

   Resizes a frame vertically. Int is amount of pixels and can be negative.

   Moveresize-specific keybinding

MoveSnap

   Snaps the frame to the closest frames or screenedges.

   Moveresize-specific keybinding

Cancel

   Cancels all moveresize actions and keeps the frame how it was before them.

   Moveresize-specific keybinding

End

   Acknowledges the moveresize actions and moves/resizes the frame as wished.

   Moveresize-specific keybinding

Menu actions:

NextItem

   Goes to next menu item.

Menu-specific keybinding

PrevItem

Goes to previous menu item.

Menu-specific keybinding

Select

Selects the current menu item.

Menu-specific keybinding

EnterSubmenu

Enters a submenu.

Menu-specific keybinding

LeaveSubmenu

Leaves a submenu.

Menu-specific keybinding

InputDialog actions:

Insert

Allows for the keypress to be inputted to the text field of InputDialog. Usually used to allow any other keys than the ones used for InputDialog.

InputDialog-specific keybinding

Erase

Erases the previous character according to the cursor position.

InputDialog-specific keybinding

Clear

Clears the whole InputDialog line.

InputDialog-specific keybinding

ClearFromCursor

Erases all characters after the current cursor position.

InputDialog-specific keybinding

Exec

Finishes input and executes the the data

Close

Closes an InputDialog.

InputDialog-specific keybinding

CursNext

Moves InputDialog cursor one characer space to right.

InputDialog-specific keybinding

CursPrev

Moves InputDialog cursor one characer space to left.

InputDialog-specific keybinding

CursEnd

Moves InputDialog cursor to the end of the line.

InputDialog-specific keybinding

CursBegin

Moves InputDialog cursor to the beginning of the line.

InputDialog-specific keybinding

HistNext

Get next history item previously used in InputDialog.

InputDialog-specific keybinding

HistPrev

Get previous history item previously used in InputDialog.

InputDialog-specific keybinding

Chapter 12. The pekwm start file

The `~/.pekwm/start` file is the simplest of all of pekwm's config files. It's a simple shell script that's run on pekwm startup. Therefore, to run, it needs to be set executable with **chmod +x ~/.pekwm/start**.

Why anyone would use `start` rather than just use their `~/.xinitrc` file? Well, the answer is, the start file is executed during the pekwm initialization phase - therefore, it gets re-executed when you issue a pekwm 'restart'.

Here's an example pekwm `start` file. Be careful to place long running applications to the background, or you will seem stuck when trying to start pekwm.

```
#!/bin/sh

xmessage 'hi. pekwm started.' &
some_command &
```

Chapter 13. Pekwm themes

This section aims to documenting the pekwm theme structure. It's rather cryptic at first look, sorry.

Please use existing themes as real life examples and base when it comes to making your own.

Guidelines

It is strongly recommended and expected that theme tarballs are labeled for the pekwm version they are made and tested with. The filename format should be
`theme_name-pekwm_version.[tar.gz|tgz|tar.bz2|tbz]`. For example
`silly_theme-pekwm_0.1.5.tar.bz2`.

It is also highly recommended that theme directories are named in a similar fashion. However, for stable releases this is not mandatory, the tarball filename is enough. If you're building for a GIT revision, mention it in as many places as possible.

The silly theme from above would contain a directory structure as follows:

```
silly_theme-pekwm_0.1.5/
pekwm_0.1.5/theme
pekwm_0.1.5/menubg.png
pekwm_0.1.5/submenu.png
```

The theme file header should contain at least the themes name, the pekwm version the theme is for, address to reach the theme maker/porter or get an updated theme, and a last modified date. Changelog entries won't hurt if you aren't the original theme author. For example:

```
# silly, a PekWM 0.1.5 theme by shared (themes@adresh.com)
# This theme is available from hewphoria.com.
# Last modified 20060529.

# Extract this theme directory under ~/.pekwm/themes/ and the
# themes menu will pick it up automatically.

# Changelog:
# 2006-05-29 HAX0ROFUNIVERSE <hawt@haxorland.invalid>
#  * REWROTE EVERYTHING WITH CAPS LOCK ON,
#    CAPS LOCK IS CRUISE CONTROL FOR COOL!
```

Try to stick to the theme syntax and rather than deleting entries please use the EMPTY texture.

Attribute names used, explanations, possible values, examples

Here is the explanation of Attributes names of themes

Attributes:

pixels

> An integer, amount of pixels.
>
> example: "2"

size

> Pixels vertically times pixels horizontally.
>
> example: "2x2"

percent

> Any percent value from 1 to 100.
>
> example: "87"

toggle

> sets a value as true (1) or false (0).
>
> example: "true"

padding

> Free pixels from top, free pixels under, free pixels from left, free pixels from right.
>
> example: "2 2 2 2"

decorname

> Name for decoration, any name can be used and applied to windows with autoproperties or the set decor action. The list below includes names with special meaning:
>
> - **DEFAULT**
>
>   Defines decorations to all windows unless overridden with another decoration set (REQUIRED).
>
> - **INPUTDIALOG**
>
>   Defines decorations for input dialogs, such as the CmdDialog.
>
> - **MENU**
>
>   Defines decorations for menus.
>
> - **STATUSWINDOW**
>
>   Defines decorations for the command dialog.
>
> - **WORKSPACEINDICATOR**
>
>   Defines decorations for the workspace indicator.
>
> - **BORDERLESS**
>
>   Defines decorations for borderless windows (recommended).
>
> - **TITLEBARLESS**
>
>   Defines decorations for titlebarless windows (recommended, should be there if your theme looks nasty when toggled titlebarless).

colour

> A colour value in RGB format.
>
> example: "#FFFFFF"

imagename

> Name of the imagefile with an option after the #
>
> - **#fixed**
>
>   Image is fixed size. Default if omitted.
>
> - **#scaled**
>
>   Image will be scaled to fit the area it's defined for.
>
> - **#tiled**
>
>   Image will be repeated as many times as needed to fill the area it's defined for.

texture

> Any valid texture. Valid textures are:
>
> - **EMPTY**
>
>   No texture (transparent).
>
> - **SOLID colour size**
>
>   A solid colour texture of defined colour and size.
>
> - **SOLIDRAISED colour colour colour pixels pixels toggle toggle toggle toggle size**
>
>   A solid colour texture with a 3D look of defined colours, form and size. First colour defines the main fill colour, second the highlight colour used on the left and top parts of the texture, third the highlight colour on the bottom and right parts of the texture. First pixel amount defines how fart apart the 3D effects are from each other, second pixel amount is how thick the bordering will be (both pixels default to 1). The four toggles are used to tell which raised corners are to be drawn. This is useful for example when defining solidraised frame corner pieces. The order is Top, Bottom, Left, Right (not unlike that used in padding). As example: "True False True False" (or 1 0 1 0) could mean you want to draw the TopLeft piece of a solidraised window border. Size should explain itself, see above.
>
> - **IMAGE imagename**
>
>   An image texture using the defined imagename

fontstring

> Defines a font. Chopped to parts by # marks. First the font name, then the text orientation, then shadow offsets, then font type if not traditional x font. Some fields can be omitted.
>
> example: "XFT#Verdana:size=10#Left#1 1" example: "-misc-fixed-*-*-*-*-14-*-*-*-*-*-*-1#Center#1 1"

buttonactions

> Buttonactions work alike what you are used from the mouse config, first mouse button number pressed when this action should happen, then any standard pekwm actions.

> example: "1" { Actions = "Close" }

Theme structure

PDecor

The block for decoration sets, any amount of Decor sections can exist inside this block.

Decor

A list of blocks with theme specifications the various types of decorations.

Title

Theming of the frame.

- Height (pixels): Amount of pixels the titlebar should height.

- HeightAdapt (boolean): If true, Height is adapted to fit the Title font.

- Pad (pixels t,l,r,b): How many pixels are left around a title text.

- Focused (texture): Background texture for a focused titlebar.

- UnFocused (texture): Background texture for an unfocused titlebar.

- WidthMin (pixels): Minimum width of title in pixels, will also place the titlebar outside of the window borders. Use 0 to place titlebar inside borders.

- WidthMax (percent): Maximum width of titles relative to window width, when this value ends up being smaller than the value in WidthMin, WidthMin is overridden.

- WidthSymetric (boolean): Set true to constant width titles or false to use titles that only are as big as the clients title text string requires (note, asymmetric width is not fully implemented yet, always set this true for now to avoid problems).

Tab

Theming of a titlebar tabs.

- Focused (texture): Background texture for a tab of a focused window.

- Unfocused (texture): Background texture for a tab of an unfocused window.

- FocusedSelected (texture): Background texture for the currently selected tab of a focused window.

- UnFocusedSelected (texture): Background texture for the currently selected tab of an unfocused window.

FontColor

Theming of font colors.

- Focused (colour colour): Text colour for a tab of a focused window. second value is the shadow colour.
- Unfocused (colour colour): Text colour for a tab of an unfocused window. second value for shadow.
- FocusedSelected (colour colour): Text colour for the currently selected tab of a focused window. second value for shadow.
- UnFocusedSelected (colour colour): Text colour for the currently selected tab of an unfocused window. second value for shadow.

Font

Theming of the titlebar fonts.

- Focused (fontstring): Font of the text of a tab of a focused window.
- Unfocused (fontstring): Font of the text of a tab of an unfocused window.
- FocusedSelected (fontstring): Font of the text of the currently selected tab of a focused window.
- UnFocusedSelected (fontstring): Font of the text of the currently selected tab of an unfocused window.

Separator

Theming of the tab separator.

- Focused (texture): Separator texture for a focused window.
- Unfocused (texture): Separator texture for an unfocused window.

Buttons

Theming of titlebar buttons.

Right = "Name"

Places the button on the right end of the titlebar.

Left = "Name"

Places the button on the left end of the titlebar.

- Focused (texture): Texture for button of a focused window.

- Unfocused (texture): Texture for button of an unfocused window.

- Pressed (texture): Texture for button that is pressed.

- Hover (texture): Texture for button when pointer is placed on it.

- Button (buttonactions): Configures what to do when a button is pressed.

Border

Theming of the borders.

Focused: borders for focused windows.

UnFocused: borders for unfocused windows.

- TopLeft (texture): Texture for the top left corner.

- Top (texture): Texture for the top border.

- TopRight (texture): Texture for the top right corner.

- Left (texture): Texture for the left border.

- Right (texture): Texture for the right birder.

- BottomLeft (texture): Texture for the bottom left corner.

- Bottom (texture): Texture for the bottom border.

- BottomRight (texture): Texture for the bottom right border.

Harbour

Enables theming of the harbour.

- Texture, texture: Texture to use as the harbour background.

Menu

Themes the insides of a menu window.

- Pad (padding): How many pixels of space around an entry is reserved.

State

One of Focused, Unfocused and Selected defining the appearance when the menu/submenu is focused, not focused and the menu entry currently selected.

- Font (fontstring): What font to use.

- Background (texture): A texture that starts from the top of the menu and ends on the bottom.

- Item (texture): A texture that starts from the top of a menu entry and ends on the bottom of the entry.

- Text (colour): Colour of text to use.

- Separator (texture): Texture to use as separator (required, client menu will break if none is defined).

- Arrow (texture): Texture to use for indicating submenus (you want this to be defined too).

CmdDialog

Themes the insides of a command dialog window.

- Font (fontstring): What font to use.

- Texture (texture): Texture to use as the background.

- Text (colour): Colour of text.

- Pad (padding): Amount of pixels of space around font to reserve.

Status

Themes the insides of the status window that shows up when moving windows and so on.

- Font (fontstring): What font to use.

- Texture (texture): Texture to use as the background.

- Text (colour): Colour of text.

- Pad (padding): Amount of pixels of space around font to reserve.

WorkspaceIndicator

Themes the workspace indicator that shows up when switching workspace.

- Font (fontstring): What font to use.
- Background (texture): Background for the whole window.
- Workspace (texture): Texture to use when rendering a workspace.
- WorkspaceActive (texture): Texture to use when rendering the active workspace.
- Text (colour): Colour of text.
- EdgePadding (padding): Amount of pixels of space around window edges and workspaces.
- WorkspacePadding (padding): Amount of pixels of space between workspaces.

Chapter 14. Mailing Lists

The pekwm mailing list (http://pekwm.org/projects/pekwm/mailing_lists/15) is used for questions, comments, development talk and announces. *You can only post to the list as a subscriber!* Please subscribe before posting using the web interface mentioned above.

Include your **pekwm --info** output with your questions and generally try to be as clear as possible so that everyone understands you.

Chapter 15. IRC

Our IRC channel #pekwm is located on irc.freenode.net. A lot of development discussion goes on there, however you can also get simple help there pretty easily.

The following is a "do not" list. If some idiotic behaviour like, public away messages, or nick changing as a way of telling what you are doing, are not mentioned it doesn't mean you can do them.

As it seems to be a problem for some, do not ask questions in IRC unless you are willing to listen to the answer. If listening is your problem, you need other help than what we at IRC can provide.

If you have a question, ask it. It doesn't help to go around shouting "can anyone help me". Help you with what? This will help the right people to take initial contact with you, saving your time and adding less clutter to the channel.

For same reasons, don't ask if we are alive or other meaningless questions leading to the real question. Don't ask if someone specific is around either. This is not the pager channel for you and your friends.

Do not ask from one person. The fact is, even how much you think he or she must be the only one who knows the answer, you are wrong. There's at least a handful of people lurking on the channel who could help you.

Read the documentation and FAQ before you ask anything. Doing this is common courtesy towards the people who answer your questions. Not reading at least the FAQ will be considered rude.

When asking a question, the answer more than often depends on the version of pekwm you use. Showing us the information the command **pekwm --info** gives will help us help you.

Do not expect people to answer your questions during the two minutes you stay in the channel. We can't sit there all the time waiting for your questions. If you can't wait longer, think about how stressful your life is, take a break, go on a vacation, take it easy for a while.

For similar reasons, don't repeat yourself. We saw it the first time. Repeating easily gets you ignored completely so it eats its own purpose. It's also considered rude, much like cutting in line.

If people tell you to read the documentation, they have a good reason to do so. If they include an URL that helps you find the info quicker, offer them your firstborn as a payback.

Don't offend anyone. They will offend you back two times.

Nicks with excessive capital letters will be hunted down and shot on sight. You might as well use a proper nick to start with to avoid this.

You can check the developers section to see the list of developers mapped to their IRC nicks.

As with all IRC channels, it's best to not do anything too drastic before keeping an eye for a while on how the channel works. We won't mind even if you just idle there, there are long standing traditions on that.

Welcome aboard!

Chapter 16. Bug Tracker

Pekwm bug tracker can be reached at pekwm.org (http://pekwm.org/projects/pekwm/tasks) . It is based on septic (https://projects.pekdon.net/projects/septic/), so bugs are inside tasks. Tasks can be reported (http://www.pekwm.org/pekwm/tasks/new) and viewed (http://www.pekwm.org/pekwm/tasks). One can even include a patch to resolve one.

Aside from bugs, tickets also eat feature requests.

Chapter 17. The developers

Below is a list of pekwm developers and what they do. The email address is more of an informational thing than anything else; the developers all subscribe to the + pekwm mailing list (http://pekwm.org/projects/pekwm/mailing_lists/15), so you should email that instead.

Developers:

Claes Nästén (aka pekdon)

> `<me@pekdon.net>`
>
> Main developer- Writes code. Has a serious screenshot fetish.

Jyri Jokinen (aka shared)

> `<shared@adresh.com>`
>
> Writes documentation. Acts ill-tempered. You have been warned.

Chapter 18. Common questions and answers

How is this ~/.pekwm/start thing used?

The file `~/.pekwm/start` is a regular shell script that is executed when pekwm starts. The file needs to be chmodded as executable (**chmod +x**) for it to get used. A simple example start file could look like this:

```
#!/bin/sh
gkrellm &
Esetroot -s mybackground.png &
```

Remember the &'s.

What is the harbour and how is it used?

Harbour is pekwm's way of supporting dockapps, special small applications that usually display things like system information or the status of your email inbox. It's essentially the same thing you might know as a dock or a wharf. The harbour is not a KDE/GNOME systray for notification icons. If you want notification icons in the harbour, you need to find a dockapp that does this for you.

The harbour needs to be enabled at compile time. When you are doing the **./configure** -phase of the pekwm compile, you can use the **--enable-harbour** option to enable it. Harbour should show up in the features list on **pekwm --info**.

If a dockapp doesn't go into the harbour even you have it enabled at compile time, you should see if the application has an option to start it "withdrawn".

Can I have automatically changing menus in pekwm?

Yes. The Dynamic keyword is a way to use automatically generated menus in pekwm. That is, menus that regenerate every time you view them. As an example, by default the themes menu is dynamic.

See Dynamic Menus for more information.

How do I install themes?

The idea is to unpack/uncompress the theme file you downloaded into some directory. In this case, we will unpack it to `~/.pekwm/themes`, which is the standard location for user installed themes.

In simple, first make sure the themes directory exist, and if not, make it by issuing the command **mkdir ~/.pekwm/themes**.

Then copy the theme package, lets call it theme.tar.gz, into `~/.pekwm/themes`. Then uncompress the theme pack with the appropriate tool. Unpack the theme with: **gzip -dc theme.tar.gz | tar xvf -**

You will then end up with a new subdirectory - this is the theme.

Since we uncompressed the theme in a standard location, after this you can select the new theme from the themes menu. If you installed in a non-standard location, you'll have to manually edit

`~/.pekwm/config`. In the top of this file there is a section named "Files {}". In this section, there is a line that says something like:

```
Theme = "/usr/local/share/pekwm/themes/minimal"
```

Edit this line to point to the directory you installed the theme. Restart pekwm and you're set.

I upgraded pekwm and now .... won't work!

Pekwm has not yet achieved a freeze on it's configuration file syntax. And as pekwm is an actively developed application, there probably have been some changes on some part of the configuration.

If you encounter a situation that when you upgrade your pekwm, and some thing just stops to work, you should either:

Move your old configuration out of the way - Move your pekwm configuration files out of `~/.pekwm` (**mv ~/.pekwm ~/old.pekwm**), which will result in new fresh configuration files being copied in. If this helps, your configuration files weren't compatible with the new version of pekwm.

Check the ChangeLog - If something configurable wise has been changed, it has also been documented in the `ChangeLog`. This is a helpful resource when you want to convert your old configuration files to a newer configuration format.

Look under the source trees `data/` *directory for reference* - If you can't find info about a new feature or for some reason you don't understand the brief explanation in the `ChangeLog`, there is a `data/` directory in the source package of pekwm that has example configuration files (that act as the default configs on a new install). Chances are you'll find help from there.

Read the documentation. - You can find links to up to date documentation for your pekwm version at the pekwm homepage.

Make sure the right executable is being executed. - Locate all instances of pekwm (**find / -name 'pekwm'**). If you see many pekwm executables laying around, maybe one in `/usr/bin` and one in `/usr/local/bin`, you might be starting a wrong version pekwm. This might happen when you for example, install a premade pekwm package for your distribution and later install pekwm from source yourself. The safe way is to remove all these pekwm instances and either re-apply the package or do **make install** again in the source. You can also, of course, go thru every pekwm binary with the --version parameter to find the right executable to keep. Note to give the full path to the executable when querying for the version (**/usr/local/bin/pekwm --version**).

Can I turn off this sloppy focus crap?

Yes. You can. You need to make all enter and leave events not affect the focus of frames, borders, clients. Simply, just comment out all the Enter lines that use the action Focus in `~/.pekwm/mouse`.

The default `~/.pekwm/mouse` configuration file has helpful "# Remove the following line if you want to use click to focus." notes in it to make this easier. Just search for such lines and remove or comment out the line (using a # in front of the line) next to the message.

See Mouse Bindings for more info on the mouse configuration file.

What is Mod1? How about Mod4?

In the `~/.pekwm/keys` and `~/.pekwm/mouse` there are all these odd Mod1 and Mod4 things used as modifier keys. It's simple - Mod1 is more widely known as the Alt key, and Mod4 as the "windows key" found on recent keyboards. Use **xev** to find out what names keys carry.

Why do my terminals start the wrong size when grouped?

This is a very complicated issue in fact, and has to do with the way terminals handle their resize actions. One way to bring at least some help to this situation is to put **resize > /dev/null** in your .bashrc or equal.

Where can I find the current size/position of a window?

Use the command **xwininfo | grep geometry**.

How do I bring up the window menu when the window has no decorations?

You press keys. The default keybinding for window menu is at the moment first Ctrl+Mod1+M, then W (or Mod4+W for short). You can specify your own keybinding for window menu at the `~/.pekwm/keys` configuration file. See Key Bindings for information on how to edit keybindings.

The start file doesn't work!

chmod +x ~/.pekwm/start. Yes, this is a duplicate of the first FAQ entry. Just making sure we never have to see this question in IRC anymore.

How do I set a background/root/desktop image?

In simple terms, you use any program that is capable of setting background images. What? Want links too? Because you asked nice, here's feh (http://www.linuxbrit.co.uk/feh/) and hsetroot (http://thegraveyard.org/hsetroot.php). If you have Eterm installed you have a program named Esetroot, that will set you backgrounds. There's a million of similar apps, and this is no place for a comprehensive list of them.

You want that the background gets set automatically when you start pekwm? Add the command for setting background in to your pekwm `start` file located at `~/.pekwm/start`. Remember to **chmod +x**.

A theme I tested doesn't work!

Pekwm is an ongoing process. This means the theme file format has gone thru considerable amounts of changes. Some of these changes are backwards compatible, some are not. You have hit a theme with too old or too new theme syntax for your copy of pekwm. Nothing can be done unless someone who knows the differences between theme formats owns you a favour and agrees to edit it out for you.

Pekwm shouldn't refuse to start again after a faulting theme test, but you will usually see everything scrambled up. In this case you can either try to select a new working theme from the menu or change the theme used manually. This is done in `~/.pekwm/config`. Under the Files-section, there is an entry named Theme, that points to your current theme. It might look something like this:

```
Files {
 Theme = "/home/shared/.pekwm/themes/blopsus9-blaah"
}
```

Now, all you need to do is make the path point to any theme you know is working. The default theme is usually a safe bet. After edited and saved, (re)start pekwm.

What desktop pagers work with pekwm?

For general use any NETWM compliant pager should do. IPager (http://www.useperl.ru/ipager/index.en.html), screenpager (http://zelea.com/project/screenpager/introduction.html), rox-pager, fbpanel's pager, obpager, gai-pager, gnome's pager, kde's pager, perlpanel's pager, netwmpager, and so on. Do report your success stories with pagers not already mentioned here.

How do I make submenus open on mouse over rather than when clicked?

You need to edit `~/.pekwm/config`. Open it in an editor and search for the Menu section towards the end of the file. It should look somewhat like this:

```
Menu {
 # Defines how menus act on mouse input.
 # Possible values are: "ButtonPress ButtonRelease DoubleClick Motion"
 # To make submenus open on mouse over, comment the default Enter,
 # uncomment the alternative, and reload pekwm.

 Select = "Motion"
 Enter = "ButtonPress"
 # Enter = "Motion"
 Exec = "ButtonRelease"
}
```

To make submenus open on mouse over, remove or comment out the line

```
    Enter = "ButtonPress"
```

and remove the # character from this line:

```
    # Enter = "Motion"
```

so that it looks like this:

```
    Enter = "Motion"
```

and you should be fine. Reload pekwm configuration from the menu or press ctrl+mod1+delete to do it with a keybinding. Test your semiautomatic pekwm menus!

The default requires you to click because of dynamic menus. While reasonably fast, they can sometimes take a second or two to be generated depending on the script behind it. Browsing the menu tree can at such times become more annoying, specially on a slower machine, than having to do that extra click. There you have it, our reasoning and the solution for if you don't like it.

My keyboard doesn't have the window keys, the default key bindings suck!

Probably the easiest way to use the default keybindings on a keyboard that has no window keys is to assign some other key the Mod4 status. Since Caps Lock is one of the most unused and annoying keys around, we'll make it the Mod4 modifier by adding the following lines to ~/.pekwm/start:

```
# Make Caps Lock act as Mod4
xmodmap -e "remove lock = Caps_Lock"
xmodmap -e "add mod4 = Caps_Lock"
```

Remember to **chmod +x ~/.pekwm/start**, then restart pekwm. Your caps lock key should now act as Mod4. Oh joy.

Where's my Unicode support?

Edit your favorite theme to use your preferred Unicode font, and enjoy all kinds of characters on the titlebars and menus.

Where did my titlebar buttons go?

Buttons require a name to be set when template based syntax is enabled in themes or the last button will be the only visible one.

Using this syntax will create only one button:

```
Require {
 Templates = "True"
}

Buttons {
 Left {
   @button
 }
 Left {
   @button
 }
}
```

Given names, both buttons will be created:

```
Require {
 Templates = "True"
```

```
}

Buttons {
 Left = "button1" {
    @button
 }
 Left = "button2" {
    @button
 }
}
```

How can I disable the workspace indicator popup?

In the main configuration file under the Screen section there should be a ShowWorkspaceIndicator parameter, set this to 0 to disable the WorkspaceIndicator.

Why is nvidia dual-head misbehaving?

The nvidia proprietary driver provides dual head information both in Xinerama and RANDR form. The RANDR information given by the driver is in the form of one large screen, the Xinerama is divided up into multiple heads. To work around this the HonourRandr option has been introduced in the Screen section of the main configuration file. Set it to False making pekwm only listen on the Xinerama information.

```
Screen {
...
    HonourRandr = "False"
...
```