Qtile Documentation

Release 0.25.1.dev16+g3e7284b.d20240423

Aldo Cortesi

Apr 23, 2024

GETTING STARTED

1 Installation 3

2 Running Qtile as a Wayland Compositor 11

3 Troubleshooting 15

4 Entry points 17

5 Default Config File 35

6 The config file 41

7 Built-in Layouts 67

8 Built-in Widgets 87

9 Built-in Hooks 181

10 Built-in Extensions 193

11 Keybindings in images 199

12 Window stacking 203

13 Architecture 205

14 Interfaces 213

15 Commands API 217

16 Hacking on Qtile 363

17 Contributing 369

18 Frequently Asked Questions 379

19 How to create a widget 385

20 How to create a layout 399

21 Using `git` 413

22 License 417

23 Changelog 419

Index 441

Qtile is a full-featured, hackable tiling window manager written and configured in Python. It's available both as an X11 window manager and also as *a Wayland compositor*.

This documentation is designed to help you *install* and *configure* Qtile. Once it's up and running you'll probably want to start adding your own *customisations* to have it running exactly the way you want.

You'll find a lot of what you need within these docs but, if you still have some questions, you can find support in the following places:

IRC
: irc://irc.oftc.net:6667/qtile

Discord
: https://discord.gg/ehh233wCrC (Bridged with IRC)

Q&A
: https://github.com/qtile/qtile/discussions/categories/q-a

Mailing List
: https://groups.google.com/group/qtile-dev

CHAPTER

ONE

INSTALLATION

1.1 Distro Guides

Below are the preferred installation methods for specific distros. If you are running something else, please see *Installing From Source*.

1.1.1 Installing on Arch Linux

Stable versions of Qtile are currently packaged for Arch Linux. To install this package, run:

```
pacman -S qtile
```

Please see the ArchWiki for more information on Qtile.

1.1.2 Installing on Fedora

Stable versions of Qtile are not currently packaged for the current version of Fedora. Users are advised to follow the instructions of *Installing From Source*.

1.1.3 Installing on Funtoo

Latest versions of Qtile are available on Funtoo. To install it, run:

```
emerge -av x11-wm/qtile
```

You can also install the development version from GitHub:

```
echo "x11-wm/qtile-9999 **" >> /etc/portage/package.accept_keywords
emerge -av qtile
```

Customize

You can customize your installation with the following useflags:

- dbus
- widget-khal-calendar
- widget-imap
- widget-keyboardkbdd
- widget-launchbar
- widget-mpd
- widget-mpris
- widget-wlan

The dbus useflag is enabled by default. Disable it only if you know what it is and know you don't use/need it.

All widget-* useflags are disabled by default because these widgets require additional dependencies while not everyone will use them. Enable only widgets you need to avoid extra dependencies thanks to these useflags.

Visit Funtoo Qtile documentation for more details on Qtile installation on Funtoo.

1.1.4 Installing on Ubuntu or Debian 11 (bullseye) or greater

Ubuntu and Debian >=11 comes with the necessary packages for installing Qtile. Starting from a minimal Debian installation, the following packages are required:

```
sudo apt install xserver-xorg xinit
sudo apt install libpangocairo-1.0-0
sudo apt install python3-pip python3-xcffib python3-cairocffi
```

Either Qtile can then be downloaded from the package index or the Github repository can be used, see *Installing From Source*:

```
pip install qtile
```

1.1.5 Installing on Slackware

Qtile is available on the SlackBuilds.org as:

Package Name	Description
qtile	stable branch (release)

Using slpkg (third party package manager)

The easy way to install Qtile is with slpkg. For example:

```
slpkg -s sbo qtile
```

Manual installation

Download dependencies first and install them. The order in which you need to install is:

- pycparser
- cffi
- futures
- python-xcffib
- trollius
- cairocffi
- qtile

Please see the HOWTO for more information on SlackBuild Usage HOWTO.

1.1.6 Installing on FreeBSD

Qtile is available via FreeBSD Ports. It can be installed with

```
pkg install qtile
```

1.1.7 Installing on NixOS

Qtile is available in the NixOS repos. To set qtile as your window manager, include this in your configuration.nix file:

```
services.xserver.windowManager.qtile.enable = true;
```

1.2 Installing From Source

1.2.1 Python interpreters

We aim to always support the last three versions of CPython, the reference Python interpreter. We usually support the latest stable version of PyPy as well. You can check the versions and interpreters we currently run our test suite against in our tox configuration file.

There are not many differences between versions aside from Python features you may or may not be able to use in your config. PyPy should be faster at runtime than any corresponding CPython version under most circumstances, especially for bits of Python code that are run many times. CPython should start up faster than PyPy and has better compatibility for external libraries.

1.2.2 Core Dependencies

Here are Qtile's core runtime dependencies and the package names that provide them in Ubuntu. Note that Qtile can run with one of two backends -- X11 and Wayland -- so only the dependencies of one of these is required.

Dependency	Ubuntu Package	Needed for
Core Dependencies		
CFFI	python3-cffi	Bars and popups
cairocffi	python3-cairocffi	Drawing on bars and popups
libpangocairo	libpangocairo-1.0-0	Writing on bars and popups
dbus-next	--	Sending notifications with dbus (optional).
X11		
X server	xserver-xorg	X11 backends
xcffib	python3-xcffib	required for X11 backend
Wayland		
wlroots	libwlroots-dev	Wayland backend (see below)
pywlroots	--	python bindings for the wlroots library
pywayland	--	python bindings for the wayland library
python-xkbcommon	--	required for wayland backeds

1.2.3 Qtile

With the dependencies in place, you can now install the stable version of qtile from PyPI:

```
pip install qtile
```

Or with sets of dependencies:

```
pip install qtile[wayland]  # for Wayland dependencies
pip install qtile[widgets]  # for all widget dependencies
pip install qtile[all]      # for all dependencies
```

Or install qtile-git with:

```
git clone https://github.com/qtile/qtile.git
cd qtile
pip install .
pip install --config-setting backend=wayland .  # adds wayland dependencies
```

1.3 Starting Qtile

There are several ways to start Qtile. The most common way is via an entry in your X session manager's menu. The default Qtile behavior can be invoked by creating a qtile.desktop file in `/usr/share/xsessions`.

A second way to start Qtile is a custom X session. This way allows you to invoke Qtile with custom arguments, and also allows you to do any setup you want (e.g. special keyboard bindings like mapping caps lock to control, setting your desktop background, etc.) before Qtile starts. If you're using an X session manager, you still may need to create a `custom.desktop` file similar to the `qtile.desktop` file above, but with `Exec=/etc/X11/xsession`. Then, create your own `~/.xsession`. There are several examples of user defined `xsession` s in the qtile-examples repository.

If there is no display manager such as SDDM, LightDM or other and there is need to start Qtile directly from `~/.xinitrc` do that by adding `exec qtile start` at the end.

In very special cases, ex. Qtile crashing during session, then suggestion would be to start through a loop to save running applications:

```
while true; do
    qtile
done
```

Finally, if you're a gnome user, you can start integrate Qtile into Gnome's session manager and use gnome as usual.

1.3.1 Running from systemd

This case will cover automatic login to Qtile after booting the system without using display manager. It logins in virtual console and init X by running through session.

Automatic login to virtual console

To get login into virtual console as an example edit *getty* service by running *systemctl edit getty@tty1* and add instructions to */etc/systemd/system/getty@tty1.service.d/override.conf*:

```
[Service]
ExecStart=
ExecStart=-/usr/bin/agetty --autologin username --noclear %I $TERM
```

username should be changed to current user name.

Check more for other examples.

Autostart X session

After login X session should be started. That can be done by *.bash_profile* if bash is used or *.zprofile* in case of zsh. Other shells can be adjusted by given examples.

```
if systemctl -q is-active graphical.target && [[ ! $DISPLAY && $XDG_VTNR -eq 1 ]]; then
  exec startx
fi
```

And to start Qtile itself *.xinitrc* should be fixed:

```
# some apps that should be started before Qtile, ex.
#
#   [[ -f ~/.Xresources ]] && xrdb -merge ~/.Xresources
#   ~/.fehbg &
#   nm-applet &
#   blueman-applet &
#   dunst &
#
# or
#
#   source ~/.xsession
```

(continues on next page)

(continued from previous page)

```
exec qtile start
```

1.3.2 Running Inside Gnome

Add the following snippet to your Qtile configuration. As per this page, it registers Qtile with gnome-session. Without it, a "Something has gone wrong!" message shows up a short while after logging in. dbus-send must be on your $PATH.

```
import subprocess
import os
from libqtile import hook

@hook.subscribe.startup
def dbus_register():
    id = os.environ.get('DESKTOP_AUTOSTART_ID')
    if not id:
        return
    subprocess.Popen(['dbus-send',
                      '--session',
                      '--print-reply',
                      '--dest=org.gnome.SessionManager',
                      '/org/gnome/SessionManager',
                      'org.gnome.SessionManager.RegisterClient',
                      'string:qtile',
                      'string:' + id])
```

This adds a new entry "Qtile GNOME" to GDM's login screen.

```
$ cat /usr/share/xsessions/qtile_gnome.desktop
[Desktop Entry]
Name=Qtile GNOME
Comment=Tiling window manager
TryExec=/usr/bin/gnome-session
Exec=gnome-session --session=qtile
Type=XSession
```

The custom session for gnome-session.

For Gnome >= 3.23.2 (Ubuntu >= 17.04, Fedora >= 26, etc.)

```
$ cat /usr/share/gnome-session/sessions/qtile.session
[GNOME Session]
Name=Qtile session
RequiredComponents=qtile;org.gnome.SettingsDaemon.A11ySettings;org.gnome.SettingsDaemon.
↪Clipboard;org.gnome.SettingsDaemon.Color;org.gnome.SettingsDaemon.Datetime;org.gnome.
↪SettingsDaemon.Housekeeping;org.gnome.SettingsDaemon.Keyboard;org.gnome.SettingsDaemon.
↪MediaKeys;org.gnome.SettingsDaemon.Mouse;org.gnome.SettingsDaemon.Power;org.gnome.
↪SettingsDaemon.PrintNotifications;org.gnome.SettingsDaemon.Rfkill;org.gnome.
↪SettingsDaemon.ScreensaverProxy;org.gnome.SettingsDaemon.Sharing;org.gnome.
↪SettingsDaemon.Smartcard;org.gnome.SettingsDaemon.Sound;org.gnome.SettingsDaemon.Wacom;
↪org.gnome.SettingsDaemon.XSettings;
```

Or for older Gnome versions

```
$ cat /usr/share/gnome-session/sessions/qtile.session
[GNOME Session]
Name=Qtile session
RequiredComponents=qtile;gnome-settings-daemon;
```

So that Qtile starts automatically on login.

```
$ cat /usr/share/applications/qtile.desktop
[Desktop Entry]
Type=Application
Encoding=UTF-8
Name=Qtile
Exec=qtile start
NoDisplay=true
X-GNOME-WMName=Qtile
X-GNOME-Autostart-Phase=WindowManager
X-GNOME-Provides=windowmanager
X-GNOME-Autostart-Notify=false
```

The above does not start gnome-panel. Getting gnome-panel to work requires some extra Qtile configuration, mainly making the top and bottom panels static on panel startup and leaving a gap at the top (and bottom) for the panel window.

You might want to add keybindings to log out of the GNOME session.

```
Key([mod, 'control'], 'l', lazy.spawn('gnome-screensaver-command -l')),
Key([mod, 'control'], 'q', lazy.spawn('gnome-session-quit --logout --no-prompt')),
Key([mod, 'shift', 'control'], 'q', lazy.spawn('gnome-session-quit --power-off')),
```

The above apps need to be in your path (though they are typically installed in `/usr/bin`, so they probably are if they're installed at all).

1.4 Wayland

Qtile can be run as a Wayland compositor rather than an X11 window manager. For this, Qtile uses wlroots, a compositor library which is undergoing fast development. Be aware that some distributions package outdated versions of wlroots. More up-to-date distributions such as Arch Linux may package pywayland, pywlroots and python-xkbcommon. Also note that we may not have yet caught up with the latest wlroots release ourselves.

Note: We currently support wlroots==0.16.0,<0.17.0 and pywlroots==0.16.4.

With the Wayland dependencies in place, Qtile can be run either from a TTY, or within an existing X11 or Wayland session where it will run inside a nested window:

```
qtile start -b wayland
```

See the *Wayland* page for more information on running Qtile as a Wayland compositor.

Similar to the xsession example above, a wayland session file can be used to start qtile from a login manager. To use this, you should create a qtile-wayland.desktop file in `/usr/share/wayland-sessions`.

1.5 udev rules

Qtile has widgets that support managing various kinds of hardware (LCD backlight, keyboard backlight, battery charge thresholds) via the kernel's exposed sysfs endpoints. However, to make this work, Qtile needs permission to write to these files. There is a udev rules file at `/resources/99-qtile.rules` in the tree, which users installing from source will want to install at `/etc/udev/rules.d/` on their system. By default, this rules file changes the group of the relevant files to the `sudo` group, and changes the file mode to be g+w (i.e. writable by all members of the sudo group). The theory here is that most systems qtile is installed on will also have the primary user in the `sudo` group. However, you can change this to whatever you like with the `--group` argument; see the sample udev rules.

Note that this file invokes Qtile's hidden `udev` from udevd, so udevd will need `qtile` in its `$PATH`. For distro packaging this shouldn't be a problem, since /usr/bin is typically in udev's path. However, for users that installed from source, you may need to modify the udev script to be one that sources your virtualenv and then invokes qtile (or just invoke it via its hardcoded path if you installed it with `--break-system-packages`), e.g.:

```
# create a wrapper script that loads the right stuff from our home directory; since
# udev will run this script as root, it has no idea about how we've installed qtile
mkdir -p $HOME/.local/bin
tee $HOME/.local/bin/qtile-udev-wrapper <<- EOF
#!/bin/sh

export PYTHONPATH=$HOME/.local/lib/python$(python3 --version | awk -F '[ .]' '{print $2
↪"." $3}')/site-packages
$HOME/.local/bin/qtile $@
EOF

# copy the in-tree udev rules file to the right place to make udev see it,
# and change the rules to point at our wrapper script above.
sed "s,qtile,$HOME/.local/bin/qtile-udev-wrapper,g" ./resources/99-qtile.rules | sudo␣
↪tee /etc/udev/rules.d/99-qtile.rules
```

CHAPTER

TWO

RUNNING QTILE AS A WAYLAND COMPOSITOR

Some functionality may not yet be implemented in the Wayland compositor. Please see the Wayland To Do List discussion for the current state of development. Also checkout the unresolved Wayland-specific issues and *troubleshooting* for tips on how to debug Wayland problems.

Note: We currently support wlroots>=0.16.0,<0.17.0 and pywlroots==0.16.4.

2.1 Backend-Specific Configuration

If you want your config file to work with different backends but want some options set differently per backend, you can check the name of the current backend in your config as follows:

```
from libqtile import qtile

if qtile.core.name == "x11":
    term = "urxvt"
elif qtile.core.name == "wayland":
    term = "foot"
```

2.2 Running X11-Only Programs

Qtile supports XWayland but requires that *wlroots* and *pywlroots* were built with XWayland support, and that XWayland is installed on the system from startup. XWayland will be started the first time it is needed.

2.2.1 XWayland windows sometimes don't receive mouse events

There is currently a known bug (https://github.com/qtile/qtile/issues/3675) which causes pointer events (hover/click/scroll) to propagate to the wrong window when switching focus.

2.3 Input Device Configuration

2.3.1 InputConfig

class libqtile.backend.wayland.**InputConfig**(***config: Any*)

This is used to configure input devices. An instance of this class represents one set of settings that can be applied to an input device.

To use this, define a dictionary called `wl_input_rules` in your config. The keys are used to match input devices, and the values are instances of this class with the desired settings. For example:

```
from libqtile.backend.wayland import InputConfig

wl_input_rules = {
    "1267:12377:ELAN1300:00 04F3:3059 Touchpad": InputConfig(left_handed=True),
    "*": InputConfig(left_handed=True, pointer_accel=True),
    "type:keyboard": InputConfig(kb_options="ctrl:nocaps,compose:ralt"),
}
```

When a input device is being configured, the most specific matching key in the dictionary is found and the corresponding settings are used to configure the device. Unique identifiers are chosen first, then `"type:X"`, then `"*"`.

The command `qtile cmd-obj -o core -f get_inputs` can be used to get information about connected devices, including their identifiers.

Options default to `None`, leave a device's default settings intact. For information on what each option does, see the documenation for libinput: https://wayland.freedesktop.org/libinput/doc/latest/configuration.html. Note that devices often only support a subset of settings.

This tries to mirror how Sway configures libinput devices. For more information check out sway-input(5): https://man.archlinux.org/man/sway-input.5#LIBINPUT_CONFIGURATION

Keyboards, managed by xkbcommon, are configured with the options prefixed by `kb_`. X11's helpful XKB guide may be useful for figuring out the syntax for some of these settings.

Configuration options

key	default	description
accel_profile	None	'adaptive' or 'flat'
click_method	None	'none', 'button_areas' or 'clickfinger'
drag	None	True or False
drag_lock	None	True or False
dwt	None	True or False
kb_layout	None	Keyboard layout i.e. XKB_DEFAULT_LAYOUT
kb_options	None	Keyboard options i.e. XKB_DEFAULT_OPTIONS
kb_repeat_delay	600	Keyboard delay in milliseconds before repeating
kb_repeat_rate	25	Keyboard key repeats made per second
kb_variant	None	Keyboard variant i.e. XKB_DEFAULT_VARIANT
left_handed	None	True or False
middle_emulation	None	True or False
natural_scroll	None	True or False
pointer_accel	None	A float between -1 and 1.
scroll_button	None	'disable', 'Button[1-3,8,9]' or a keycode
scroll_method	None	'none', 'two_finger', 'edge', or 'on_button_down'
tap	None	True or False
tap_button_map	None	'lrm' or 'lmr'

If you want to change keyboard configuration during runtime, you can use the core's *set_keymap* command (see below).

2.4 Core Commands

See the *Wayland backend* section in the API Commands documentation.

CHAPTER

THREE

TROUBLESHOOTING

3.1 So something has gone wrong... what do you do?

When Qtile is running, it logs error messages (and other messages) to its log file. This is found at `~/.local/share/qtile/qtile.log`. This is the first place to check to see what is going on. If you are getting unexpected errors from normal usage or your configuration (and you're not doing something wacky) and believe you have found a bug, then please *report a bug*.

If you are *hacking on Qtile* and you want to debug your changes, this log is your best friend. You can send messages to the log from within libqtile by using the `logger`:

```
from libqtile.log_utils import logger

logger.warning("Your message here")
logger.warning(variable_you_want_to_print)

try:
    # some changes here that might error
except Exception:
    logger.exception("Uh oh!")
```

`logger.warning` is convenient because its messages will always be visibile in the log. `logger.exception` is helpful because it will print the full traceback of an error to the log. By sticking these amongst your changes you can look more closely at the effects of any changes you made to Qtile's internals.

3.2 X11: Capturing an `xtrace`

Occasionally, a bug will be low level enough to require an `xtrace` of Qtile's conversations with the X server. To capture one of these, create an `xinitrc` or similar file with:

```
exec xtrace qtile >> ~/qtile.log
```

This will put the xtrace output in Qtile's logfile as well. You can then demonstrate the bug, and paste the contents of this file into the bug report.

Note that xtrace may be named `x11trace` on some platforms, for example, on Fedora.

3.3 Debugging in Wayland

To get incredibly verbose output of communications between clients and the server, you can set `WAYLAND_DEBUG=1` in the environment before starting the process. This applies to the server itself, so be aware that running `qtile` with this set will generate lots of output for Qtile **and** all clients that it launches. If you're including this output with a bug report please try to cut out just the relevant portions.

If you're hacking on Qtile and would like this debug log output for it rather than any clients, it can be helpful to run the helper script at `scripts/wephyr` in the source from an existing session. You can then run clients from another terminal using the `WAYLAND_DISPLAY` value printed by Qtile, so that the debug logs printed by Qtile are only the server's.

If you suspect a client may be responsible for a bug, it can be helpful to look at the issue trackers for other compositors, such as sway. Similarly if you're hacking on Qtile's internals and think you've found an unexpected quirk it may be helpful to search the issue tracker for wlroots.

CHAPTER

FOUR

ENTRY POINTS

Qtile uses a subcommand structure; various subcommands are listed below. Additionally, two other commands available in the scripts/ section of the repository are also documented below.

4.1 qtile start

This is the entry point for the window manager, and what you should run from your `.xsession` or similar. This will make an attempt to detect if qtile is already running and fail if it is. See `qtile start --help` for more details.

4.2 qtile shell

The Qtile command shell is a command-line shell interface that provides access to the full complement of Qtile command functions. The shell features command name completion, and full command documentation can be accessed from the shell itself. The shell uses GNU Readline when it's available, so the interface can be configured to, for example, obey VI keybindings with an appropriate `.inputrc` file. See the GNU Readline documentation for more information.

4.2.1 Navigating the Object Graph

The shell presents a filesystem-like interface to the command graph - the builtin "cd" and "ls" commands act like their familiar shell counterparts:

```
> ls
layout/  widget/  screen/  bar/     window/  group/

> cd screen
layout/  window/  bar/  widget/

> cd ..
/

> ls
layout/  widget/  screen/  bar/     window/  group/
```

If you try to access an object that has no "default" value then you will see an error message:

```
> ls
layout/  widget/  screen/  bar/     window/  group/
```

(continues on next page)

(continued from previous page)

```
> cd bar
Item required for bar

> ls bar
bar[bottom]/

> cd bar/bottom
bar['bottom']> ls
screen/  widget/
```

Please refer to *Navigating the command graph* for a summary of which objects need a specified selector and the type of selector required. Using `ls` will show which selectors are available for an object. Please see below for an explanation about how Qtile displays shell paths.

Alternatively, the `items()` command can be run on the parent object to show which selectors are available. The first value shows whether a selector is optional (`False` means that a selector is required) and the second value is a list of selectors:

```
> ls
layout/  widget/  screen/  bar/     window/  group/

> items(bar)
(False, ['bottom'])
```

4.2.2 Displaying the shell path

Note that the shell provides a "short-hand" for specifying node keys (as opposed to children). The following is a valid shell path:

```
> cd group/4/window/31457314
```

The command prompt will, however, always display the Python node path that should be used in scripts and key bindings:

```
group['4'].window[31457314]>
```

4.2.3 Live Documentation

The shell `help` command provides the canonical documentation for the Qtile API:

```
> cd layout/1

layout[1]> help
help command   -- Help for a specific command.

Builtins
========
cd    exit  help  ls    q     quit

Commands for this object
```

(continues on next page)

(continued from previous page)

```
=======================
add             commands        current         delete          doc
down            get_info        items           next            previous
rotate          shuffle_down    shuffle_up      toggle_split    up

layout[1]> help previous
previous()
Focus previous stack.
```

4.3 qtile migrate

`qtile migrate` is a tool to help users update their configs to reflect any breaking changes/deprecations introduced in later versions.

The tool can automatically apply updates but it can also be used to highlight impacted lines, allowing users to update their configs manually.

The tool can take a number of options when running:

Argument	Description	Default
`-c, --config`	Sets the path to the config file	`~/.config/qtile/config.py`
`--list-m`	Lists all the available migrations that can be run by the tool.	n/a
`--info ID`	Show more detail about the migration implement by ID.	n/a
`--after-VERSION`	Only runs migrations relating to changes implemented after release VERSION.	Not set (i.e. runs all migrations).
`-r ID, --run-mi ID`	Run selected migrations identified by ID. Comma separated list if using multiple values.	Not set (i.e. runs all migrations).
`--yes`	Automatically apply changes without asking user for confirmation.	Not set (i.e. users will need to confirm application of changes).
`--show-d`	When used with `--yes` will cause diffs to still be shown for information purposes only.	Not set.
`--no-col`	Disables colour output for diff.	Not set
`--lint`	Outputs linting lines showing location of changes. No changes are made to the config.	Not set.

4.3.1 Available migrations

The following migrations are currently available.

ID	Changes introduced after version	Summary
UpdateBitcoin	0.18.0	Updates `BitcoinTicker` to `CryptoTicker`.
UpdateBluetoothArgs	0.23.0	Updates `Bluetooth` argument signature.
UpdateKeychordArgs	0.21.0	Updates `KeyChord` argument signature.
UpdateStocktickerArgs	0.22.1	Updates `StockTicker` argument signature.
UpdateWidgetboxArgs	0.20.0	Updates `WidgetBox` argument signature.
MatchListRegex	0.23.0	Updates Match objects using lists
ModuleRenames	0.18.1	Updates certain deprecated `libqtile.` module names.
RemoveCmdPrefix	0.22.1	Removes `cmd_` prefix from method calls and definitions.
RenamePacmanWidget	0.16.1	Changes deprecated `Pacman` widget name to `CheckUpdates`.
RenameWindowNameHook	0.16.1	Changes `window_name_changed` hook name.
RenameThreadedPollText	0.16.1	Replaces `ThreadedPollText` with `ThreadPoolText`.
RenameTileMaster	0.16.1	Changes `masterWindows` argument to `master_length`.
UpdateMonadArgs	0.17.0	Updates `new_at_current` keyword argument in Monad layouts.
UpdateTogroupArgs	0.18.1	Updates `groupName` keyword argument to `group_name`.

4.3.2 Running migrations

Assuming your config file is in the default location, running `qtile migrate` is sufficent to start the migration process.

Let's say you had a config file with the following contents:

```
import libqtile.command_client

keys = [
    KeyChord(
        [mod],
        "x",
        [Key([], "Up", lazy.layout.grow()), Key([], "Down", lazy.layout.shrink())],
        mode="Resize layout",
    )
]

qtile.cmd_spawn("alacritty")
```

Running `qtile migrate` will run each available migration and, where the migration would result in changes, a diff will be shown and you will be asked whether you wish to apply the changes.

```
UpdateKeychordArgs: Updates ``KeyChord`` argument signature.

--- original
+++ modified
@@ -5,7 +5,8 @@
```

(continues on next page)

(continued from previous page)

```
        [mod],
        "x",
        [Key([], "Up", lazy.layout.grow()), Key([], "Down", lazy.layout.shrink())],
-        mode="Resize layout",
+        name="Resize layout",
+    mode=True,
    )
]

Apply changes? (y)es, (n)o, (s)kip file, (q)uit.
```

You will see from the output above that you are shown the name of the migration being applied and its purpose, along with the changes that will be implemented.

If you select `quit` the migration will be stopped and any applied changes will be reversed.

Once all migrations have been run on a file, you will then be asked whether you want to save changes to the file:

```
Save all changes to config.py? (y)es, (n)o.
```

At the end of the migration, backups of your original config will still be in your config folder. NB these will be overwritten if you re-run `qtile migrate`.

4.3.3 Linting

If you don't want the script to modify your config directly, you can use the `--lint` option to show you where changes are required.

Running `qtile migrate --lint` on the same config as shown above will result in the following output:

```
config.py:
[Ln 1, Col 7]: The 'libqtile.command_*' modules have been moved to 'libqtile.command.*'.␣
↪(ModuleRenames)
[Ln 8, Col 8]: The use of mode='mode name' for KeyChord is deprecated. Use mode=True and␣
↪value='mode name'. (UpdateKeychordArgs)
[Ln 12, Col 6]: Use of 'cmd_' prefix is deprecated. 'cmd_spawn' should be replaced with
↪'spawn' (RemoveCmdPrefix)
```

4.3.4 Explanations of migrations

The table below provides more detail of the available migrations.

UpdateBitcoin

Migration introduced after version	0.18.0

The `BitcoinTicker` widget has been renamed `CryptoTicker`. In addition, the `format` keyword argument is removed during this migration as the available fields for the format have changed.

The removal only happens on instances of `BitcoinTracker`. i.e. running `qtile migrate` on the following code:

```
BitcoinTicker(format="...")
CryptoTicker(format="...")
```

will return:

```
CryptoTicker()
CryptoTicker(format="...")
```

UpdateBluetoothArgs

Migration introduced after version	0.23.0

The `Bluetooth` widget previously accepted a `hci` keyword argument. This has been deprecated following a major overhaul of the widget and should be replaced with a keyword argument named `device`.

For example:

```
widget.Bluetooth(hci="/dev_XX_XX_XX_XX_XX_XX")
```

should be changed to:

```
widget.Bluetooth(device="/dev_XX_XX_XX_XX_XX_XX")
```

UpdateKeychordArgs

Migration introduced after version	0.21.0

Previously, users could make a key chord persist by setting the *mode* to a string representing the name of the mode. For example:

```
keys = [
    KeyChord(
        [mod],
        "x",
        [
            Key([], "Up", lazy.layout.grow()),
            Key([], "Down", lazy.layout.shrink())
        ],
        mode="Resize layout",
```

(continues on next page)

(continued from previous page)

```
    )
]
```

This will now result in the following warning message in the log file:

```
The use of `mode` to set the KeyChord name is deprecated. Please use `name='Resize Layout
↪'` instead.
'mode' should be a boolean value to set whether the chord is persistent (True) or not."
```

To remove the error, the config should be amended as follows:

```
keys = [
    KeyChord(
        [mod],
        "x",
        [
            Key([], "Up", lazy.layout.grow()),
            Key([], "Down", lazy.layout.shrink())
        ],
        name="Resize layout",
    mode=True,
    )
]
```

Note: The formatting of the inserted argument may not correctly match your own formatting. You may this to run a tool like `black` after applying this migration to tidy up your code.

UpdateStocktickerArgs

Migration introduced after version	0.22.1

The `StockTicker` widget had a keyword argument called `function`. This needs to be renamed to `func` to prevent clashes with the `function()` method of `CommandObject`.

For example:

```
widget.StockTicker(function="TIME_SERIES_INTRADAY")
```

should be changed to:

```
widget.StockTicker(func="TIME_SERIES_INTRADAY")
```

UpdateWidgetboxArgs

Migration introduced after version	0.20.0

The `WidgetBox` widget allowed a position argument to set the contents of the widget. This behaviour is deprecated and, instead, the contents should be specified with a keyword argument called `widgets`.

For example:

```
widget.WidgetBox(
    [
        widget.Systray(),
        widget.Volume(),
    ]
)
```

should be changed to:

```
widget.WidgetBox(
    widgets=[
        widget.Systray(),
        widget.Volume(),
    ]
)
```

MatchListRegex

Migration introduced after version	0.23.0

The use of lists in `Match` objects is deprecated and should be replaced with a regex.

For example:

```
Match(wm_class=["one", "two"])
```

should be changed to:

```
Match(wm_class=re.compile(r"^(one|two)$"))
```

ModuleRenames

Migration introduced after version	0.18.1

To tidy up the qtile codebase, the `libqtile.command_*` modules were moved to `libqtile.command.*` with one exception, `libqtile.command_object` was renamed `libqtile.command.base`.

In addition, the `libqtile.window` module was moved to `libqtile.backend.x11.window`.

NB. this migration will update imports in the following forms:

```
import libqtile.command_client
import libqtile.command_interface as interface
from libqtile import command_client
from libqtile.command_client import CommandClient
```

Results in:

```
import libqtile.command.client
import libqtile.command.interface as interface
from libqtile.command.client import CommandClient
from libqtile.command import client
```

RemoveCmdPrefix

Migration introduced after version	0.22.1

The cmd_ prefix was used to identify methods that should be exposed to qtile's command API. This has been deprecated and so calls no longer require the prefix.

For example:

```
qtile.cmd_spawn("vlc")
```

would be replaced with:

```
qtile.spawn("vlc")
```

Where users have created their own widgets with methods using this prefix, the syntax has also changed:

For example:

```
class MyWidget(libqtile.widget.base._Widget):
    def cmd_my_command(self):
        pass
```

Should be updated as follows:

```
from libqtile.command.base import expose_command

class MyWidget(libqtile.widget.base._Widget):
    @expose_command
    def my_command(self):
        pass
```

RenamePacmanWidget

Migration introduced after version	0.16.1

The `Pacman` widget has been renamed to `CheckUpdates`.

This is because the widget supports multiple package managers.

Example:

```
screens = [
    Screen(
        top=Bar(
            [
              ...
              widget.Pacman(),
              ...
            ]
        )
    )
]
```

Should be updated as follows:

```
screens = [
    Screen(
        top=Bar(
            [
              ...
              widget.CheckUpdates(),
              ...
            ]
        )
    )
]
```

RenameWindowNameHook

Migration introduced after version	0.16.1

The `window_name_changed` hook has been replaced with `client_name_updated`.

Example:

```
@hook.subscribe.window_name_changed
def my_func(window):
    ...
```

Should be updated as follows:

```
@hook.subscribe.client_name_updated
def my_func(window):
    ...
```

RenameThreadedPollText

Migration introduced after version	0.16.1

The `ThreadedPollText` class needs to replced with `ThreadPoolText`.

This is because the `ThreadPoolText` class can do everything that the `ThreadedPollText` does so the redundant code was removed.

Example:

```
from libqtile import widget

class MyPollingWidget(widget.base.ThreadedPollText):
    ...
```

Should be updated as follows:

```
from libqtile import widget

class MyPollingWidget(widget.base.ThreadPoolText):
    ...
```

RenameTileMaster

Migration introduced after version	0.16.1

To be consistent with other layouts, the `masterWindows` property of the `Tile` layout was renamed to `master_length`. Configs using the `masterWindows` argument when configuring the layout should replace this.

UpdateMonadArgs

Migration introduced after version	0.17.0

Replaces the `new_at_current=True|False` argument in `Monad*` layouts with `new_client_position` to be consistent with other layouts.

`new_at_current=True` is replaced with `new_client_position="before_current` and `new_at_current=False` is replaced with `new_client_position="after_current"`.

UpdateTogroupArgs

Migration introduced after version	0.18.1

To be consistent with codestyle, the `groupName` argument in the `togroup` command needs to be changed to `group_name`.

The following code:

```
lazy.window.togroup(groupName="1")
```

will result in a warning in your logfile: `Window.togroup's groupName is deprecated; use group_name`.

The code should be updated to:

```
lazy.window.togroup(group_name="1")
```

4.4 qtile cmd-obj

This is a simple tool to expose qtile.command functionality to shell. This can be used standalone or in other shell scripts.

4.4.1 How it works

`qtile cmd-obj` works by selecting a command object and calling a specified function of that object.

As per *Architecture*, Qtile's command graph has seven nodes: `layout`, `window`, `group`, `bar`, `widget`, `screen`, and a special `root` node. These are the objects that can be accessed via `qtile cmd-obj`.

Running the command against a selected object without a function (`-f`) will run the `help` command and list the commands available to the object. Commands shown with an asterisk ("*") require arguments to be passed via the `-a` flag.

Selecting an object

With the exception of `cmd`, all objects need an identifier so the correct object can be selected. Refer to *Navigating the command graph* for more information.

Note: You will see from the graph on *Architecture* that certain objects can be accessed from other objects. For example, `qtile cmd-obj -o group term layout` will list the commands for the current layout on the `term` group.

Information on functions

Running a function with the `-i` flag will provide additional detail about that function (i.e. what it does and what arguments it expects).

Passing arguments to functions

Arguments can be passed to a function by using the `-a` flag. For example, to change the label for the group named "1" to "A", you would run `qtile cmd-obj -o group 1 -f set_label -a A`.

Warning: It is not currently possible to pass non-string arguments to functions via `qtile cmd-obj`. Doing so will result in an error.

4.4.2 Examples:

Output of `qtile cmd-obj -h`

```
usage: qtile cmd-obj [-h] [--object OBJ_SPEC [OBJ_SPEC ...]]
                 [--function FUNCTION] [--args ARGS [ARGS ...]] [--info]

Simple tool to expose qtile.command functionality to shell.

optional arguments:
  -h, --help            show this help message and exit
  --object OBJ_SPEC [OBJ_SPEC ...], -o OBJ_SPEC [OBJ_SPEC ...]
                        Specify path to object (space separated). If no
                        --function flag display available commands.
  --function FUNCTION, -f FUNCTION
                        Select function to execute.
  --args ARGS [ARGS ...], -a ARGS [ARGS ...]
                        Set arguments supplied to function.
  --info, -i            With both --object and --function args prints
                        documentation for function.

Examples:
 qtile cmd-obj
 qtile cmd-obj -o root  # same as above, root node is default
 qtile cmd-obj -o root -f prev_layout -i
 qtile cmd-obj -o root -f prev_layout -a 3 # prev_layout on group 3
 qtile cmd-obj -o group 3 -f focus_back
 qtile cmd-obj -o widget textbox -f update -a "New text"
 qtile cmd-obj -f restart # restart qtile
```

Output of `qtile cmd-obj -o group 3`

```
-o group 3 -f commands              Returns a list of possible commands for this object
-o group 3 -f doc                 * Returns the documentation for a specified command name
-o group 3 -f eval                * Evaluates code in the same context as this function
-o group 3 -f focus_back            Focus the window that had focus before the current one␣
↪got it.
-o group 3 -f focus_by_name       * Focus the first window with the given name. Do nothing␣
↪if the name is
-o group 3 -f function            * Call a function with current object as argument
-o group 3 -f info                  Returns a dictionary of info for this group
-o group 3 -f info_by_name        * Get the info for the first window with the given name␣
↪without giving it
-o group 3 -f items               * Returns a list of contained items for the specified␣
↪name
-o group 3 -f next_window           Focus the next window in group.
-o group 3 -f prev_window           Focus the previous window in group.
-o group 3 -f set_label           * Set the display name of current group to be used in␣
↪GroupBox widget.
-o group 3 -f setlayout
-o group 3 -f switch_groups       * Switch position of current group with name
-o group 3 -f toscreen            * Pull a group to a specified screen.
-o group 3 -f unminimize_all        Unminimise all windows in this group
```

Output of `qtile cmd-obj -o root`

```
-o root -f add_rule               * Add a dgroup rule, returns rule_id needed to remove it
-o root -f addgroup               * Add a group with the given name
-o root -f commands                 Returns a list of possible commands for this object
-o root -f critical                 Set log level to CRITICAL
-o root -f debug                    Set log level to DEBUG
-o root -f delgroup               * Delete a group with the given name
-o root -f display_kb             * Display table of key bindings
-o root -f doc                    * Returns the documentation for a specified command name
-o root -f error                    Set log level to ERROR
-o root -f eval                   * Evaluates code in the same context as this function
-o root -f findwindow             * Launch prompt widget to find a window of the given␣
↪name
-o root -f focus_by_click         * Bring a window to the front
-o root -f function               * Call a function with current object as argument
-o root -f get_info                 Prints info for all groups
-o root -f get_state                Get pickled state for restarting qtile
-o root -f get_test_data            Returns any content arbitrarily set in the self.test_
↪data attribute.
-o root -f groups                   Return a dictionary containing information for all␣
↪groups
-o root -f hide_show_bar          * Toggle visibility of a given bar
-o root -f info                     Set log level to INFO
-o root -f internal_windows         Return info for each internal window (bars, for␣
↪example)
-o root -f items                  * Returns a list of contained items for the specified␣
```

(continues on next page)

(continued from previous page)

```
↪name
-o root -f list_widgets            List of all addressible widget names
-o root -f next_layout           * Switch to the next layout.
-o root -f next_screen             Move to next screen
-o root -f next_urgent             Focus next window with urgent hint
-o root -f pause                   Drops into pdb
-o root -f prev_layout           * Switch to the previous layout.
-o root -f prev_screen             Move to the previous screen
-o root -f qtile_info              Returns a dictionary of info on the Qtile instance
-o root -f qtilecmd              * Execute a Qtile command using the client syntax
-o root -f remove_rule           * Remove a dgroup rule by rule_id
-o root -f restart                 Restart qtile
-o root -f run_extension         * Run extensions
-o root -f run_external          * Run external Python script
-o root -f screens                 Return a list of dictionaries providing information␣
↪on all screens
-o root -f shutdown                Quit Qtile
-o root -f simulate_keypress     * Simulates a keypress on the focused window.
-o root -f spawn                 * Run cmd in a shell.
-o root -f spawncmd              * Spawn a command using a prompt widget, with tab-
↪completion.
-o root -f status                  Return "OK" if Qtile is running
-o root -f switch_groups         * Switch position of groupa to groupb
-o root -f switchgroup           * Launch prompt widget to switch to a given group to␣
↪the current screen
-o root -f sync                    Sync the X display. Should only be used for␣
↪development
-o root -f to_layout_index       * Switch to the layout with the given index in self.
↪layouts.
-o root -f to_screen             * Warp focus to screen n, where n is a 0-based screen␣
↪number
-o root -f togroup               * Launch prompt widget to move current window to a␣
↪given group
-o root -f tracemalloc_dump        Dump tracemalloc snapshot
-o root -f tracemalloc_toggle      Toggle tracemalloc status
-o root -f warning                 Set log level to WARNING
-o root -f windows                 Return info for each client window
```

4.5 qtile run-cmd

Run a command applying rules to the new windows, ie, you can start a window in a specific group, make it floating, intrusive, etc.

The Windows must have NET_WM_PID.

```
# run xterm floating on group "test-group"
qtile run-cmd -g test-group -f xterm
```

4.6 qtile top

`qtile top` is a `top`-like tool to measure memory usage of Qtile's internals.

Note: To use `qtile shell` you need to have `tracemalloc` enabled. You can do this by setting the environmental variable `PYTHONTRACEMALLOC=1` before starting qtile. Alternatively, you can force start `tracemalloc` but you will lose early traces:

```
>>> from libqtile.command.client import InteractiveCommandClient
>>> i=InteractiveCommandClient()
>>> i.eval("import tracemalloc;tracemalloc.start()")
```

4.7 dqtile-cmd

A Rofi/dmenu interface to qtile-cmd. Accepts all arguments of qtile-cmd.

4.7.1 Examples:

Output of `dqtile-cmd -o cmd`

```
dmenu:                                                                                                -
Alt-1   Prompt for args and show function help (if -f is present)
..      Go back to menu.
C-u     Clear input
Esc     Exit
-o cmd -f add_rule              * Add a dgroup rule, returns rule_id needed to remove it
-o cmd -f addgroup              * Add a group with the given name
-o cmd -f commands                Returns a list of possible commands for this object
-o cmd -f critical                Set log level to CRITICAL
-o cmd -f debug                   Set log level to DEBUG
-o cmd -f delgroup              * Delete a group with the given name
-o cmd -f display_kb            * Display table of key bindings
-o cmd -f doc                   * Returns the documentation for a specified command name
-o cmd -f error                   Set log level to ERROR
-o cmd -f eval                  * Evaluates code in the same context as this function
-o cmd -f findwindow            * Launch prompt widget to find a window of the given name
-o cmd -f focus_by_click        * Bring a window to the front
-o cmd -f function              * Call a function with current object as argument
-o cmd -f get_info                Prints info for all groups
-o cmd -f get_state               Get pickled state for restarting qtile
```

Output of `dqtile-cmd -h`

```
dqtile-cmd

    A Rofi/dmenu interface to qtile-cmd. Excepts all arguments of qtile-cmd
    (see below).

usage: dqtile-cmd [-h] [--object OBJ_SPEC [OBJ_SPEC ...]]
                  [--function FUNCTION] [--args ARGS [ARGS ...]] [--info]

Simple tool to expose qtile.command functionality to shell.

optional arguments:
  -h, --help            show this help message and exit
  --object OBJ_SPEC [OBJ_SPEC ...], -o OBJ_SPEC [OBJ_SPEC ...]
                        Specify path to object (space separated). If no
                        --function flag display available commands.
  --function FUNCTION, -f FUNCTION
                        Select function to execute.
  --args ARGS [ARGS ...], -a ARGS [ARGS ...]
                        Set arguments supplied to function.
  --info, -i            With both --object and --function args prints
                        documentation for function.

Examples:
 dqtile-cmd
 dqtile-cmd -o cmd
 dqtile-cmd -o cmd -f prev_layout -i
 dqtile-cmd -o cmd -f prev_layout -a 3 # prev_layout on group 3
 dqtile-cmd -o group 3 -f focus_back

If both rofi and dmenu are present rofi will be selected as default, to change this us --
↪force-dmenu as the first argument.
```

4.8 iqshell

In addition to the standard `qtile shell` shell interface, we provide a kernel capable of running through Jupyter that hooks into the qshell client. The command structure and syntax is the same as qshell, so it is recommended you read that for more information about that.

4.8.1 Dependencies

In order to run iqshell, you must have ipykernel and jupyter_console. You can install the dependencies when you are installing qtile by running:

```
$ pip install qtile[ipython]
```

Otherwise, you can just install these two packages separately, either through PyPI or through your distribution package manager.

4.8.2 Installing and Running the Kernel

Once you have the required dependencies, you can run the kernel right away by running:

```
$ python3 -m libqtile.interactive.iqshell_kernel
```

However, this will merely spawn a kernel instance, you will have to run a separate frontend that connects to this kernel.

A more convenient way to run the kernel is by registering the kernel with Jupyter. To register the kernel itself, run:

```
$ python3 -m libqtile.interactive.iqshell_install
```

If you run this as a non-root user, or pass the `--user` flag, this will install to the user Jupyter kernel directory. You can now invoke the kernel directly when starting a Jupyter frontend, for example:

```
$ jupyter console --kernel qshell
```

The `iqshell` script will launch a Jupyter terminal console with the qshell kernel.

4.8.3 iqshell vs qtile shell

One of the main drawbacks of running through a Jupyter kernel is the frontend has no way to query the current node of the kernel, and as such, there is no way to set a custom prompt. In order to query your current node, you can call `pwd`.

This, however, enables many of the benefits of running in a Jupyter frontend, including being able to save, run, and re-run code cells in frontends such as the Jupyter notebook.

The Jupyter kernel also enables more advanced help, text completion, and introspection capabilities (however, these are currently not implemented at a level much beyond what is available in the standard qtile shell).

CHAPTER

FIVE

DEFAULT CONFIG FILE

The below default config file is included with the Qtile package and will be copied to your home config folder (~/.config/qtile/config.py) if no config file exists when you start Qtile for the first time.

```
# Copyright (c) 2010 Aldo Cortesi
# Copyright (c) 2010, 2014 dequis
# Copyright (c) 2012 Randall Ma
# Copyright (c) 2012-2014 Tycho Andersen
# Copyright (c) 2012 Craig Barnes
# Copyright (c) 2013 horsik
# Copyright (c) 2013 Tao Sauvage
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.

from libqtile import bar, layout, qtile, widget
from libqtile.config import Click, Drag, Group, Key, Match, Screen
from libqtile.lazy import lazy
from libqtile.utils import guess_terminal

mod = "mod4"
terminal = guess_terminal()

keys = [
    # A list of available commands that can be bound to keys can be found
    # at https://docs.qtile.org/en/latest/manual/config/lazy.html
```

(continues on next page)

(continued from previous page)

```
    # Switch between windows
    Key([mod], "h", lazy.layout.left(), desc="Move focus to left"),
    Key([mod], "l", lazy.layout.right(), desc="Move focus to right"),
    Key([mod], "j", lazy.layout.down(), desc="Move focus down"),
    Key([mod], "k", lazy.layout.up(), desc="Move focus up"),
    Key([mod], "space", lazy.layout.next(), desc="Move window focus to other window"),
    # Move windows between left/right columns or move up/down in current stack.
    # Moving out of range in Columns layout will create new column.
    Key([mod, "shift"], "h", lazy.layout.shuffle_left(), desc="Move window to the left"),
    Key([mod, "shift"], "l", lazy.layout.shuffle_right(), desc="Move window to the right
↪"),
    Key([mod, "shift"], "j", lazy.layout.shuffle_down(), desc="Move window down"),
    Key([mod, "shift"], "k", lazy.layout.shuffle_up(), desc="Move window up"),
    # Grow windows. If current window is on the edge of screen and direction
    # will be to screen edge - window would shrink.
    Key([mod, "control"], "h", lazy.layout.grow_left(), desc="Grow window to the left"),
    Key([mod, "control"], "l", lazy.layout.grow_right(), desc="Grow window to the right
↪"),
    Key([mod, "control"], "j", lazy.layout.grow_down(), desc="Grow window down"),
    Key([mod, "control"], "k", lazy.layout.grow_up(), desc="Grow window up"),
    Key([mod], "n", lazy.layout.normalize(), desc="Reset all window sizes"),
    # Toggle between split and unsplit sides of stack.
    # Split = all windows displayed
    # Unsplit = 1 window displayed, like Max layout, but still with
    # multiple stack panes
    Key(
        [mod, "shift"],
        "Return",
        lazy.layout.toggle_split(),
        desc="Toggle between split and unsplit sides of stack",
    ),
    Key([mod], "Return", lazy.spawn(terminal), desc="Launch terminal"),
    # Toggle between different layouts as defined below
    Key([mod], "Tab", lazy.next_layout(), desc="Toggle between layouts"),
    Key([mod], "w", lazy.window.kill(), desc="Kill focused window"),
    Key(
        [mod],
        "f",
        lazy.window.toggle_fullscreen(),
        desc="Toggle fullscreen on the focused window",
    ),
    Key([mod], "t", lazy.window.toggle_floating(), desc="Toggle floating on the focused␣
↪window"),
    Key([mod, "control"], "r", lazy.reload_config(), desc="Reload the config"),
    Key([mod, "control"], "q", lazy.shutdown(), desc="Shutdown Qtile"),
    Key([mod], "r", lazy.spawncmd(), desc="Spawn a command using a prompt widget"),
]

# Add key bindings to switch VTs in Wayland.
# We can't check qtile.core.name in default config as it is loaded before qtile is␣
↪started
# We therefore defer the check until the key binding is run by using .when(func=...)
```

(continues on next page)

(continued from previous page)

```
for vt in range(1, 8):
    keys.append(
        Key(
            ["control", "mod1"],
            f"f{vt}",
            lazy.core.change_vt(vt).when(func=lambda: qtile.core.name == "wayland"),
            desc=f"Switch to VT{vt}",
        )
    )


groups = [Group(i) for i in "123456789"]

for i in groups:
    keys.extend(
        [
            # mod + group number = switch to group
            Key(
                [mod],
                i.name,
                lazy.group[i.name].toscreen(),
                desc="Switch to group {}".format(i.name),
            ),
            # mod + shift + group number = switch to & move focused window to group
            Key(
                [mod, "shift"],
                i.name,
                lazy.window.togroup(i.name, switch_group=True),
                desc="Switch to & move focused window to group {}".format(i.name),
            ),
            # Or, use below if you prefer not to switch to that group.
            # # mod + shift + group number = move focused window to group
            # Key([mod, "shift"], i.name, lazy.window.togroup(i.name),
            #     desc="move focused window to group {}".format(i.name)),
        ]
    )

layouts = [
    layout.Columns(border_focus_stack=["#d75f5f", "#8f3d3d"], border_width=4),
    layout.Max(),
    # Try more layouts by unleashing below layouts.
    # layout.Stack(num_stacks=2),
    # layout.Bsp(),
    # layout.Matrix(),
    # layout.MonadTall(),
    # layout.MonadWide(),
    # layout.RatioTile(),
    # layout.Tile(),
    # layout.TreeTab(),
    # layout.VerticalTile(),
    # layout.Zoomy(),
]
```

(continues on next page)

(continued from previous page)

```
widget_defaults = dict(
    font="sans",
    fontsize=12,
    padding=3,
)
extension_defaults = widget_defaults.copy()

screens = [
    Screen(
        bottom=bar.Bar(
            [
                widget.CurrentLayout(),
                widget.GroupBox(),
                widget.Prompt(),
                widget.WindowName(),
                widget.Chord(
                    chords_colors={
                        "launch": ("#ff0000", "#ffffff"),
                    },
                    name_transform=lambda name: name.upper(),
                ),
                widget.TextBox("default config", name="default"),
                widget.TextBox("Press &lt;M-r&gt; to spawn", foreground="#d75f5f"),
                # NB Systray is incompatible with Wayland, consider using StatusNotifier␣
↪instead
                # widget.StatusNotifier(),
                widget.Systray(),
                widget.Clock(format="%Y-%m-%d %a %I:%M %p"),
                widget.QuickExit(),
            ],
            24,
            # border_width=[2, 0, 2, 0],  # Draw top and bottom borders
            # border_color=["ff00ff", "000000", "ff00ff", "000000"]  # Borders are␣
↪magenta
        ),
        # You can uncomment this variable if you see that on X11 floating resize/moving␣
↪is laggy
        # By default we handle these events delayed to already improve performance,␣
↪however your system might still be struggling
        # This variable is set to None (no cap) by default, but you can set it to 60 to␣
↪indicate that you limit it to 60 events per second
        # x11_drag_polling_rate = 60,
    ),
]

# Drag floating layouts.
mouse = [
    Drag([mod], "Button1", lazy.window.set_position_floating(), start=lazy.window.get_
↪position()),
    Drag([mod], "Button3", lazy.window.set_size_floating(), start=lazy.window.get_
↪size()),
```

(continues on next page)

(continued from previous page)

```
    Click([mod], "Button2", lazy.window.bring_to_front()),
]

dgroups_key_binder = None
dgroups_app_rules = []  # type: list
follow_mouse_focus = True
bring_front_click = False
floats_kept_above = True
cursor_warp = False
floating_layout = layout.Floating(
    float_rules=[
        # Run the utility of `xprop` to see the wm class and name of an X client.
        *layout.Floating.default_float_rules,
        Match(wm_class="confirmreset"),  # gitk
        Match(wm_class="makebranch"),  # gitk
        Match(wm_class="maketag"),  # gitk
        Match(wm_class="ssh-askpass"),  # ssh-askpass
        Match(title="branchdialog"),  # gitk
        Match(title="pinentry"),  # GPG key password entry
    ]
)
auto_fullscreen = True
focus_on_window_activation = "smart"
reconfigure_screens = True

# If things like steam games want to auto-minimize themselves when losing
# focus, should we respect this or not?
auto_minimize = True

# When using the Wayland backend, this can be used to configure input devices.
wl_input_rules = None

# XXX: Gasp! We're lying here. In fact, nobody really uses or cares about this
# string besides java UI toolkits; you can see several discussions on the
# mailing lists, GitHub issues, and other WM documentation that suggest setting
# this string if your java app doesn't work correctly. We may as well just lie
# and say that we're a working one by default.
#
# We choose LG3D to maximize irony: it is a 3D non-reparenting WM written in
# java that happens to be on java's whitelist.
wmname = "LG3D"
```

CHAPTER

SIX

THE CONFIG FILE

Qtile is configured in Python. A script (`~/.config/qtile/config.py` by default) is evaluated, and a small set of configuration variables are pulled from its global namespace.

6.1 Configuration lookup order

Qtile looks in the following places for a configuration file, in order:

- The location specified by the `-c` argument.
- `$XDG_CONFIG_HOME/qtile/config.py`, if it is set
- `~/.config/qtile/config.py`
- first `qtile/config.py` found in `$XDG_CONFIG_DIRS` (defaults to `/etc/xdg`)
- It reads the module `libqtile.resources.default_config`, included by default with every Qtile installation.

Qtile will try to create the configuration file as a copy of the default config, if it doesn't exist yet, this one will be placed inside of `$XDG_CONFIG_HOME/qtile/config.py` (if set) or `~/.config/qtile/config.py`.

6.2 Default Configuration

The *default configuration* is invoked when qtile cannot find a configuration file. In addition, if qtile is restarted or the config is reloaded, qtile will load the default configuration if the config file it finds has some kind of error in it. The documentation below describes the configuration lookup process, as well as what the key bindings are in the default config.

The default config is not intended to be suitable for all users; it's mostly just there so qtile does /something/ when fired up, and so that it doesn't crash and cause you to lose all your work if you reload a bad config.

6.3 Configuration variables

A Qtile configuration consists of a file with a bunch of variables in it, which qtile imports and then runs as a Python file to derive its final configuration. The documentation below describes the most common configuration variables; more advanced configuration can be found in the qtile-examples repository, which includes a number of real-world configurations that demonstrate how you can tune Qtile to your liking. (Feel free to issue a pull request to add your own configuration to the mix!)

6.3.1 Lazy objects

Lazy objects are a way of executing any of the commands available in Qtile's *commands API*.

The name "lazy" refers to the fact that the commands are not executed at the time of the call. Instead, the lazy object creates a reference to the relevant command and this is only executed when the relevant event is triggered (e.g. on a keypress).

Typically, for config files, the commands are used to manipulate windows, layouts and groups as well application commands like exiting, restarting, reloading the config file etc.

Example

```
from libqtile.config import Key
from libqtile.lazy import lazy

keys = [
    Key(
        ["mod1"], "k",
        lazy.layout.down()
    ),
    Key(
        ["mod1"], "j",
        lazy.layout.up()
    )
]
```

Note: As noted above, `lazy` calls do not call the relevant command but only create a reference to it. While this makes it ideal for binding commands to key presses and `mouse_callbacks` for widgets, it also means that `lazy` calls cannot be included in user-defined functions.

Lazy functions

This is overview of the commonly used functions for the key bindings. These functions can be called from commands on the REPLACE object or on another object in the command tree.

Some examples are given below. For a complete list of available commands, please refer to *Commands API*.

General functions

function	description
`lazy.spawn("applicatic`	Run the `application`
`lazy.spawncmd()`	Open command prompt on the bar. See prompt widget.
`lazy.reload_config()`	Reload the config.
`lazy.restart()`	Restart Qtile. In X11, it won't close your windows.
`lazy.shutdown()`	Close the whole Qtile

Group functions

function	description
`lazy.next_layout()`	Use next layout on the actual group
`lazy.prev_layout()`	Use previous layout on the actual group
`lazy.screen.next_group()`	Move to the group on the right
`lazy.screen.prev_group()`	Move to the group on the left
`lazy.screen.toggle_group()`	Move to the last visited group
`lazy.group.next_window()`	Switch window focus to next window in group
`lazy.group.prev_window()`	Switch window focus to previous window in group
`lazy.group["group_name"].toscreen()`	Move to the group called `group_name`. Takes an optional `toggle` parameter (defaults to False). If this group is already on the screen, it does nothing by default; to toggle with the last used group instead, use `toggle=True`.
`lazy.layout.increase_ratio()`	Increase the space for master window at the expense of slave windows
`lazy.layout.decrease_ratio()`	Decrease the space for master window in the advantage of slave windows

Window functions

function	description
`lazy.window.kill()`	Close the focused window
`lazy.layout.next()`	Switch window focus to other pane(s) of stack
`lazy.window.togroup("group_na`	Move focused window to the group called `group_name`
`lazy.window.toggle_floating()`	Put the focused window to/from floating mode
`lazy.window.toggle_fullscreen`	Put the focused window to/from fullscreen mode
`lazy.window.move_up()`	Move the window above the next window in the stack.
`lazy.window.move_down()`	Move the window below the previous window in the stack.
`lazy.window.move_to_top()`	Move the window above all other windows with similar priority (i.e. a "normal" window will not be moved above a `kept_above` window).
`lazy.window.move_to_bottom()`	Move the window below all other windows with similar priority (i.e. a "normal" window will not be moved below a `kept_below` window).
`lazy.window.keep_above()`	Keep window above other windows.
`lazy.window.keep_below()`	Keep window below other windows.
`lazy.window.bring_to_front()`	Bring window above all other windows. Ignores `kept_above` priority.

Screen functions

function	description
`lazy.screen.set_wallpaper(pat mode=None)`	Set the wallpaper to the specificied image. Possible modes: `None` no resizing, `'fill'` centre and resize to fill screen, `'stretch'` stretch to fill screen.

ScratchPad DropDown functions

function	description
`lazy.group["group_name dropdown_toggle("`	Toggles the visibility of the specified DropDown window. On first use, the configured process is spawned.
`lazy.group["group_name hide_all()`	Hides all DropDown windows.
`lazy.group["group_name dropdown_reconfig **configuration)`	Update the configuration of the named DropDown.

User-defined functions

function	description
`lazy.function(func, *args, **kwargs)`	Calls `func(qtile, *args, **kwargs)`. NB. the `qtile` object is automatically passed as the first argument.

Examples

`lazy.function` can also be used as a decorator for functions.

```
from libqtile.config import Key
from libqtile.lazy import lazy

@lazy.function
def my_function(qtile):
    ...

keys = [
    Key(
        ["mod1"], "k",
        my_function
    )
]
```

Additionally, you can pass arguments to user-defined function in one of two ways:

1) In-line definition

Arguments can be added to the `lazy.function` call.

```
from libqtile.config import Key
from libqtile.lazy import lazy
```

(continues on next page)

(continued from previous page)

```
from libqtile.log_utils import logger

def multiply(qtile, value, multiplier=10):
    logger.warning(f"Multiplication results: {value * multiplier}")

keys = [
    Key(
        ["mod1"], "k",
        lazy.function(multiply, 10, multiplier=2)
    )
]
```

2) Decorator

Arguments can also be passed to the decorated function.

```
from libqtile.config import Key
from libqtile.lazy import lazy
from libqtile.log_utils import logger

@lazy.function
def multiply(qtile, value, multiplier=10):
    logger.warning(f"Multiplication results: {value * multiplier}")

keys = [
    Key(
        ["mod1"], "k",
        multiply(10, multiplier=2)
    )
]
```

6.3.2 Groups

A group is a container for a bunch of windows, analogous to workspaces in other window managers. Each client window managed by the window manager belongs to exactly one group. The `groups` config file variable should be initialized to a list of `Group` objects.

`Group` objects provide several options for group configuration. Groups can be configured to show and hide themselves when they're not empty, spawn applications for them when they start, automatically acquire certain groups, and various other options.

Example

```
from libqtile.config import Group, Match

groups = [
    Group("a"),
    Group("b"),
    Group("c", matches=[Match(wm_class="Firefox")]),
]
```

(continues on next page)

(continued from previous page)

```
# allow mod3+1 through mod3+0 to bind to groups; if you bind your groups
# by hand in your config, you don't need to do this.
from libqtile.dgroups import simple_key_binder

dgroups_key_binder = simple_key_binder("mod3")
```

Reference

Group

class libqtile.config.**Group**(*name: str*, *matches: list[Match] | None = None*, *exclusive: bool = False*, *spawn: str | list[str] | None = None*, *layout: str | None = None*, *layouts: list[Layout] | None = None*, *persist: bool = True*, *init: bool = True*, *layout_opts: dict[str, Any] | None = None*, *screen_affinity: int | None = None*, *position: int = 9223372036854775807*, *label: str | None = None*)

Represents a "dynamic" group

These groups can spawn apps, only allow certain Matched windows to be on them, hide when they're not in use, etc. Groups are identified by their name.

Parameters

name:
The name of this group.

matches:
List of `Match` objects whose matched windows will be assigned to this group.

exclusive:
When other apps are started in this group, should we allow them here or not?

spawn:
This will be executed (via `qtile.spawn()`) when the group is created. You can pass either a program name or a list of programs to `exec()`.

layout:
The name of default layout for this group (e.g. `"max"`). This is the name specified for a particular layout in `config.py` or if not defined it defaults in general to the class name in all lower case.

layouts:
The group layouts list overriding global layouts. Use this to define a separate list of layouts for this particular group.

persist:
Should this group stay alive when it has no member windows?

init:
Should this group be alive when Qtile starts?

layout_opts:
Options to pass to a layout.

screen_affinity:
Make a dynamic group prefer to start on a specific screen.

position:
The position of this group.

label:
The display name of the group. Use this to define a display name other than name of the group. If set to `None`, the display name is set to the name.

`libqtile.dgroups.`**`simple_key_binder`**(*mod*, *keynames=None*)

Bind keys to mod+group position or to the keys specified as second argument

Group Matching

Match

`class` `libqtile.config.`**`Match`**(*title: str | re.Pattern | None = None*, *wm_class: str | re.Pattern | None = None*, *role: str | re.Pattern | None = None*, *wm_type: str | re.Pattern | None = None*, *wm_instance_class: str | re.Pattern | None = None*, *net_wm_pid: int | None = None*, *func: Callable[[base.Window], bool] | None = None*, *wid: int | None = None*)

Window properties to compare (match) with a window.

The properties will be compared to a `Window` to determine if its properties *match*. It can match by title, wm_class, role, wm_type, wm_instance_class, net_wm_pid, or wid. Additionally, a function may be passed, which takes in the `Window` to be compared against and returns a boolean.

For some properties, `Match` supports both regular expression objects (i.e. the result of `re.compile()`) or strings (match as an exact string). If a window matches all specified values, it is considered a match.

Parameters

title:
Match against the WM_NAME atom (X11) or title (Wayland).

wm_class:
Match against any value in the whole WM_CLASS atom (X11) or app ID (Wayland).

role:
Match against the WM_ROLE atom (X11 only).

wm_type:
Match against the WM_TYPE atom (X11 only).

wm_instance_class:
Match against the first string in WM_CLASS atom (X11) or app ID (Wayland).

net_wm_pid:
Match against the _NET_WM_PID atom (X11) or PID (Wayland).

func:
Delegate the match to the given function, which receives the tested client as an argument and must return `True` if it matches, `False` otherwise.

wid:
Match against the window ID. This is a unique ID given to each window.

Rule

class libqtile.config.**Rule**(*match: Match | list[Match]*, *group: _Group | None = None*, *float: bool = False*, *intrusive: bool = False*, *break_on_match: bool = True*)

How to act on a match.

A `Rule` contains a list of `Match` objects, and a specification about what to do when any of them is matched.

Parameters

match:
`Match` object or a list of such associated with this rule.

float:
Should we auto float this window?

intrusive:
Should we override the group's exclusive setting?

break_on_match:
Should we stop applying rules if this rule is matched?

ScratchPad and DropDown

`ScratchPad` is a special - by default invisible - group which acts as a container for `DropDown` configurations. A `DropDown` can be configured to spawn a defined process and bind thats process' window to it. The associated window can then be shown and hidden by the lazy command `dropdown_toggle()` (see *Lazy objects*) from the ScratchPad group. Thus - for example - your favorite terminal emulator turns into a quake-like terminal by the control of Qtile.

If the DropDown window turns visible it is placed as a floating window on top of the current group. If the DropDown is hidden, it is simply switched back to the ScratchPad group.

Example

```
from libqtile.config import Group, ScratchPad, DropDown, Key
from libqtile.lazy import lazy

groups = [
    ScratchPad("scratchpad", [
        # define a drop down terminal.
        # it is placed in the upper third of screen by default.
        DropDown("term", "urxvt", opacity=0.8),

        # define another terminal exclusively for ``qtile shell` at different position
        DropDown("qtile shell", "urxvt -hold -e 'qtile shell'",
                 x=0.05, y=0.4, width=0.9, height=0.6, opacity=0.9,
                 on_focus_lost_hide=True) ]),
    Group("a"),
]

keys = [
  # toggle visibiliy of above defined DropDown named "term"
  Key([], 'F11', lazy.group['scratchpad'].dropdown_toggle('term')),
```

(continues on next page)

(continued from previous page)

```
  Key([], 'F12', lazy.group['scratchpad'].dropdown_toggle('qtile shell')),
]
```

Note that if the window is set to not floating, it is detached from DropDown and ScratchPad, and a new process is spawned next time the DropDown is set visible.

Some programs run in a server-like mode where the spawned process does not directly own the window that is created, which is instead created by a background process. In this case, the window may not be correctly caught in the scratchpad group. To work around this, you can pass a `Match` object to the corresponding `DropDown`. See below.

Reference

ScratchPad

class libqtile.config.**ScratchPad**(*name: str*, *dropdowns: list[libqtile.config.DropDown] | None = None*, *position: int = 9223372036854775807*, *label: str = ''*, *single: bool = False*)

Represents a "ScratchPad" group

ScratchPad adds a (by default) invisible group to Qtile. That group is used as a place for currently not visible windows spawned by a `DropDown` configuration.

Parameters

name:
The name of this group.

dropdowns:
`DropDown` s available on the scratchpad.

position:
The position of this group.

label:
The display name of the `ScratchPad` group. Defaults to the empty string such that the group is hidden in `GroupBox` widget.

single:
If `True`, only one of the dropdowns will be visible at a time.

DropDown

class libqtile.config.**DropDown**(*name: str*, *cmd: str*, ***config: Any*)

Configure a specified command and its associated window for the `ScratchPad`. That window can be shown and hidden using a configurable keystroke or any other scripted trigger.

Configuration options

key	default	description
height	0.35	Height of window as fraction of current screen.
match	None	Use a Match to identify the spawned window and move it to the scratchpad, instead of relying on the window's PID. This works around some programs that may not be caught by the window's PID if it does not match the PID of the spawned process.
on_focus_lost_hi	True	Shall the window be hidden if focus is lost? If so, the DropDown is hidden if window focus or the group is changed.
opacity	0.9	Opacity of window as fraction. One is opaque.
warp_pointer	True	Shall pointer warp to center of window on activation? This only has effect if any of the on_focus_lost_xxx options are True
width	0.8	Width of window as fraction of current screen width
x	0.1	X position of window as fraction of current screen width. 0 is the left most position.
y	0.0	Y position of window as fraction of current screen height. 0 is the top most position. To show the window at bottom, you have to configure a value < 1 and an appropriate height.

6.3.3 Keys

The keys variable defines Qtile's key bindings.

Default Key Bindings

The mod key for the default config is mod4, which is typically bound to the "Super" keys, which are things like the windows key and the mac command key. The basic operation is:

- mod + k or mod + j: switch windows on the current stack
- mod + <space>: put focus on the other pane of the stack (when in stack layout)
- mod + <tab>: switch layouts
- mod + w: close window
- mod + <ctrl> + r: reload the config
- mod + <group name>: switch to that group
- mod + <shift> + <group name>: send a window to that group
- mod + <enter>: start terminal guessed by libqtile.utils.guess_terminal
- mod + r: start a little prompt in the bar so users can run arbitrary commands

The default config defines one screen and 8 groups, one for each letter in asdfuiop. It has a basic bottom bar that includes a group box, the current window name, a little text reminder that you're using the default config, a system tray, and a clock.

The default configuration has several more advanced key combinations, but the above should be enough for basic usage of qtile.

See *Keybindings in images* for visual keybindings in keyboard layout.

Defining key bindings

Individual key bindings are defined with `Key` as demonstrated in the following example. Note that you may specify more than one callback functions.

```
from libqtile.config import Key

keys = [
    # Pressing "Meta + Shift + a".
    Key(["mod4", "shift"], "a", callback, ...),

    # Pressing "Control + p".
    Key(["control"], "p", callback, ...),

    # Pressing "Meta + Tab".
    Key(["mod4", "mod1"], "Tab", callback, ...),
]
```

The above may also be written more concisely with the help of the `EzKey` helper class. The following example is functionally equivalent to the above:

```
from libqtile.config import EzKey as Key

keys = [
    Key("M-S-a", callback, ...),
    Key("C-p",   callback, ...),
    Key("M-A-<Tab>", callback, ...),
]
```

The `EzKey` modifier keys (i.e. `MASC`) can be overwritten through the `EzKey.modifier_keys` dictionary. The defaults are:

```
modifier_keys = {
    'M': 'mod4',
    'A': 'mod1',
    'S': 'shift',
    'C': 'control',
}
```

Callbacks can also be configured to work only under certain conditions by using the `when()` method. Currently, the following conditions are supported:

```
from libqtile.config import Key

keys = [
    # Only trigger callback for a specific layout
    Key(
        [mod, 'shift'],
        "j",
        lazy.layout.grow().when(layout='verticaltile'),
        lazy.layout.grow_down().when(layout='columns')
    ),

    # Limit action to when the current window is not floating (default True)
```

(continues on next page)

(continued from previous page)

```
    Key([mod], "f", lazy.window.toggle_fullscreen().when(when_floating=False))

    # Also matches are supported on the current window
    # For example to match on the wm_class for fullscreen do the following
    Key([mod], "f", lazy.window.toggle_fullscreen().when(focused=Match(wm_class=
↪"yourclasshere"))
]
```

KeyChords

Qtile also allows sequences of keys to trigger callbacks. These sequences are known as chords and are defined with `KeyChord`. Chords are added to the `keys` section of the config file.

```
from libqtile.config import Key, KeyChord

keys = [
    KeyChord([mod], "z", [
        Key([], "x", lazy.spawn("xterm"))
    ])
]
```

The above code will launch xterm when the user presses Mod + z, followed by x.

Warning: Users should note that key chords are aborted by pressing <escape>. In the above example, if the user presses Mod + z, any following key presses will still be sent to the currently focussed window. If <escape> has not been pressed, the next press of x will launch xterm.

Modes

Chords can optionally persist until a user presses <escape>. This can be done by setting `mode=True`. This can be useful for configuring a subset of commands for a particular situations (i.e. similar to vim modes).

```
from libqtile.config import Key, KeyChord

keys = [
    KeyChord([mod], "z", [
        Key([], "g", lazy.layout.grow()),
        Key([], "s", lazy.layout.shrink()),
        Key([], "n", lazy.layout.normalize()),
        Key([], "m", lazy.layout.maximize())],
        mode=True,
        name="Windows"
    )
]
```

In the above example, pressing Mod + z triggers the "Windows" mode. Users can then resize windows by just pressing g (to grow the window), s to shrink it etc. as many times as needed. To exit the mode, press <escape>.

Note: The Chord widget (`Chord`) will display the name of the active chord (as set by the `name` parameter). This is particularly useful where the chord is a persistent mode as this will indicate when the chord's mode is still active.

Chains

Chords can also be chained to make even longer sequences.

```
from libqtile.config import Key, KeyChord

keys = [
    KeyChord([mod], "z", [
        KeyChord([], "x", [
            Key([], "c", lazy.spawn("xterm"))
        ])
    ])
]
```

Modes can also be added to chains if required. The following example demonstrates the behaviour when using the `mode` argument in chains:

```
from libqtile.config import Key, KeyChord

keys = [
    KeyChord([mod], "z", [
        KeyChord([], "y", [
            KeyChord([], "x", [
                Key([], "c", lazy.spawn("xterm"))
            ], mode=True, name="inner")
        ])
    ], mode=True, name="outer")
]
```

After pressing Mod+z y x c, the "inner" mode will remain active. When pressing <escape>, the "inner" mode is exited. Since the mode in between does not have `mode` set, it is also left. Arriving at the "outer" mode (which has this argument set) stops the "leave" action and "outer" now becomes the active mode.

Note: If you want to bind a custom key to leave the current mode (e.g. Control + G in addition to `<escape>`), you can specify `lazy.ungrab_chord()` as the key action. To leave all modes and return to the root bindings, use `lazy.ungrab_all_chords()`.

Modifiers

On most systems `mod1` is the Alt key - you can see which modifiers, which are enclosed in a list, map to which keys on your system by running the `xmodmap` command. This example binds `Alt-k` to the "down" command on the current layout. This command is standard on all the included layouts, and switches to the next window (where "next" is defined differently in different layouts). The matching "up" command switches to the previous window.

Modifiers include: "shift", "lock", "control", "mod1", "mod2", "mod3", "mod4", and "mod5". They can be used in combination by appending more than one modifier to the list:

```
Key(
    ["mod1", "control"], "k",
    lazy.layout.shuffle_down()
)
```

Special keys

These are most commonly used special keys. For complete list please see the code. You can create bindings on them just like for the regular keys. For example `Key(["mod1"], "F4", lazy.window.kill())`.

Return
BackSpace
Tab
space
Home, End
Left, Up, Right, Down
F1, F2, F3, ...
XF86AudioRaiseVolume
XF86AudioLowerVolume
XF86AudioMute
XF86AudioNext
XF86AudioPrev
XF86MonBrightnessUp
XF86MonBrightnessDown

Reference

Key

class libqtile.config.**Key**(*modifiers: list[str]*, *key: str*, **commands: LazyCall*, *desc: str = ''*, *swallow: bool = True*)

Defines a keybinding.

Parameters

modifiers:
A list of modifier specifications. Modifier specifications are one of: `"shift"`, `"lock"`, `"control"`, `"mod1"`, `"mod2"`, `"mod3"`, `"mod4"`, `"mod5"`.

key:
A key specification, e.g. `"a"`, `"Tab"`, `"Return"`, `"space"`.

commands:
One or more `LazyCall` objects to evaluate in sequence upon keypress. Multiple commands should be separated by commas.

desc:
Description to be added to the key binding. (Optional)

swallow:
Configures when we swallow the key binding. (Optional) Setting it to False will forward the key binding to the focused window after the commands have been executed.

KeyChord

`class libqtile.config.KeyChord(`*modifiers: list[str]*, *key: str*, *submappings: list[libqtile.config.Key | libqtile.config.KeyChord]*, *mode: bool | str = False*, *name: str = ''*, *desc: str = ''*, *swallow: bool = True*`)`

Define a key chord aka Vim-like mode.

Parameters

modifiers:
A list of modifier specifications. Modifier specifications are one of: `"shift"`, `"lock"`, `"control"`, `"mod1"`, `"mod2"`, `"mod3"`, `"mod4"`, `"mod5"`.

key:
A key specification, e.g. `"a"`, `"Tab"`, `"Return"`, `"space"`.

submappings:
A list of `Key` or `KeyChord` declarations to bind in this chord.

mode:
Boolean. Setting to `True` will result in the chord persisting until Escape is pressed. Setting to `False` (default) will exit the chord once the sequence has ended.

name:
A string to name the chord. The name will be displayed in the Chord widget.

desc:
A string to describe the chord. This attribute is not directly used by Qtile but users may want to access this when creating scripts to show configured keybindings.

swallow:
Configures when we swallow the key binding of the chord. (Optional) Setting it to False will forward the key binding to the focused window after the commands have been executed.

EzKey

`class libqtile.config.EzKey(`*keydef: str*, **commands: LazyCall*, *desc: str = ''*`)`

Defines a keybinding using the Emacs-like format.

Parameters

keydef:
The Emacs-like key specification, e.g. `"M-S-a"`.

commands:
A list `LazyCall` objects to evaluate in sequence upon keypress.

desc:
Description to be added to the key binding. (Optional)

6.3.4 Layouts

A layout is an algorithm for laying out windows in a group on your screen. Since Qtile is a tiling window manager, this usually means that we try to use space as efficiently as possible, and give the user ample commands that can be bound to keys to interact with layouts.

The `layouts` variable defines the list of layouts you will use with Qtile. The first layout in the list is the default. If you define more than one layout, you will probably also want to define key bindings to let you switch to the next and previous layouts.

See *Built-in Layouts* for a listing of available layouts.

Example

```
from libqtile import layout

layouts = [
    layout.Max(),
    layout.Stack(stacks=2)
]
```

6.3.5 Mouse

The `mouse` config file variable defines a set of global mouse actions, and is a list of `Click` and `Drag` objects, which define what to do when a window is clicked or dragged.

Default Mouse Bindings

By default, holding your `mod` key and left-clicking (and holding) a window will allow you to drag it around as a floating window. Holding your `mod` key and right-clicking (and holding) a window will resize the window (and also make it float if it is not already floating).

Example

```
from libqtile.config import Click, Drag
mouse = [
    Drag([mod], "Button1", lazy.window.set_position_floating(),
        start=lazy.window.get_position()),
    Drag([mod], "Button3", lazy.window.set_size_floating(),
        start=lazy.window.get_size()),
    Click([mod], "Button2", lazy.window.bring_to_front())
]
```

The above example can also be written more concisely with the help of the `EzClick` and `EzDrag` helpers:

```
from libqtile.config import EzClick as Click, EzDrag as Drag

mouse = [
    Drag("M-1", lazy.window.set_position_floating(),
        start=lazy.window.get_position()),
    Drag("M-3", lazy.window.set_size_floating(),
        start=lazy.window.get_size()),
    Click("M-2", lazy.window.bring_to_front())
]
```

Reference

Click

class libqtile.config.**Click**(*modifiers: list[str]*, *button: str*, **commands: LazyCall*)

Bind commands to a clicking action.

Parameters

modifiers:
A list of modifier specifications. Modifier specifications are one of: `"shift"`, `"lock"`, `"control"`, `"mod1"`, `"mod2"`, `"mod3"`, `"mod4"`, `"mod5"`.

button:
The button used to start dragging e.g. `"Button1"`.

commands:
A list `LazyCall` objects to evaluate in sequence upon drag.

Drag

class libqtile.config.**Drag**(*modifiers: list[str]*, *button: str*, **commands: LazyCall*, *start: LazyCall | None = None*, *warp_pointer: bool = False*)

Bind commands to a dragging action.

On each motion event the bound commands are executed with two additional parameters specifying the x and y offset from the previous position.

Parameters

modifiers:
A list of modifier specifications. Modifier specifications are one of: `"shift"`, `"lock"`, `"control"`, `"mod1"`, `"mod2"`, `"mod3"`, `"mod4"`, `"mod5"`.

button:
The button used to start dragging e.g. `"Button1"`.

commands:
A list `LazyCall` objects to evaluate in sequence upon drag.

start:
A `LazyCall` object to be evaluated when dragging begins. (Optional)

warp_pointer:
A `bool` indicating if the pointer should be warped to the bottom right of the window at the start of dragging. (Default: *False*)

EzClick

class libqtile.config.**EzClick**(*btndef: str*, **commands: LazyCall*)

Bind commands to a clicking action using the Emacs-like format.

Parameters

btndef:
The Emacs-like button specification, e.g. `"M-1"`.

commands:
A list `LazyCall` objects to evaluate in sequence upon drag.

6.3.6 Screens

The `screens` configuration variable is where the physical screens, their associated `bars`, and the `widgets` contained within the bars are defined (see *Built-in Widgets* for a listing of available widgets).

Example

Tying together screens, bars and widgets, we get something like this:

```
from libqtile.config import Screen
from libqtile import bar, widget

window_name = widget.WindowName()

screens = [
    Screen(
        bottom=bar.Bar([
            widget.GroupBox(),
            window_name,
            ], 30),
        ),
    Screen(
        bottom=bar.Bar([
            widget.GroupBox(),
            window_name,
            ], 30),
        )
    ]
```

Note that a widget can be passed to multiple bars (and likewise multiple times to the same bar). Its contents is mirrored across all copies so this is useful where you want identical content (e.g. the name of the focussed window, like in this example).

Bars support both solid background colors and gradients by supplying a list of colors that make up a linear gradient. For example, `bar.Bar(..., background="#000000")` will give you a black back ground (the default), while `bar.Bar(..., background=["#000000", "#FFFFFF"])` will give you a background that fades from black to white.

Bars (and widgets) also support transparency by adding an alpha value to the desired color. For example, `bar.Bar(..., background="#00000000")` will result in a fully transparent bar. Widget contents will not be impacted i.e. this is different to the `opacity` parameter which sets the transparency of the entire window.

Note: In X11 backends, transparency will be disabled in a bar if the `background` color is fully opaque.

Users can add borders to the bar by using the `border_width` and `border_color` parameters. Providing a single value sets the value for all four sides while sides can be customised individually by setting four values in a list (top, right, bottom, left) e.g. `border_width=[2, 0, 2, 0]` would draw a border 2 pixels thick on the top and bottom of the bar.

Multiple Screens

You will see from the example above that `screens` is a list of individual `Screen` objects. The order of the screens in this list should match the order of screens as seen by your display server.

X11

You can view the current order of your screens by running `xrandr --listmonitors`.

Examples of how to set the order of your screens can be found on the Arch wiki.

Wayland

The Wayland backend supports the wlr-output-management protocol for configuration of outputs by tools such as Kanshi.

Fake Screens

instead of using the variable *screens* the variable *fake_screens* can be used to set split a physical monitor into multiple screens. They can be used like this:

```
from libqtile.config import Screen
from libqtile import bar, widget

# screens look like this
#     600          300
#  |-------------|-----|
#  |          480|     |580
#  |   A         |  B  |
#  |----------|--|     |
#  |       400|--|-----|
#  |   C      |        |400
#  |----------|   D    |
#     500     |--------|
#                 400
#
# Notice there is a hole in the middle
# also D goes down below the others

fake_screens = [
  Screen(
      bottom=bar.Bar(
          [
```

(continues on next page)

(continued from previous page)

```
            widget.Prompt(),
            widget.Sep(),
            widget.WindowName(),
            widget.Sep(),
            widget.Systray(),
            widget.Sep(),
            widget.Clock(format='%H:%M:%S %d.%m.%Y')
        ],
        24,
        background="#555555"
    ),
    x=0,
    y=0,
    width=600,
    height=480
),
Screen(
    top=bar.Bar(
        [
            widget.GroupBox(),
            widget.WindowName(),
            widget.Clock()
        ],
        30,
    ),
    x=600,
    y=0,
    width=300,
    height=580
),
Screen(
    top=bar.Bar(
        [
            widget.GroupBox(),
            widget.WindowName(),
            widget.Clock()
        ],
        30,
    ),
    x=0,
    y=480,
    width=500,
    height=400
),
Screen(
    top=bar.Bar(
        [
            widget.GroupBox(),
            widget.WindowName(),
            widget.Clock()
        ],
        30,
```

(continues on next page)

(continued from previous page)

```
        ),
        x=500,
        y=580,
        width=400,
        height=400
    ),
]
```

Third-party bars

There might be some reasons to use third-party bars. For instance you can come from another window manager and you have already configured dzen2, xmobar, or something else. They definitely can be used with Qtile too. In fact, any additional configurations aren't needed. Just run the bar and qtile will adapt.

Reference

Screen

class libqtile.config.**Screen**(**args*, ***kwargs*)

A physical screen, and its associated paraphernalia.

Define a screen with a given set of `Bar`s of a specific geometry. Also, `x`, `y`, `width`, and `height` aren't specified usually unless you are using 'fake screens'.

The `wallpaper` parameter, if given, should be a path to an image file. How this image is painted to the screen is specified by the `wallpaper_mode` parameter. By default, the image will be placed at the screens origin and retain its own dimensions. If the mode is `"fill"`, the image will be centred on the screen and resized to fill it. If the mode is `"stretch"`, the image is stretched to fit all of it into the screen.

The `x11_drag_polling_rate` parameter specifies the rate for drag events in the X11 backend. By default this is set to None, indicating no limit. Because in the X11 backend we already handle motion notify events later, the performance should already be okay. However, to limit these events further you can use this variable and e.g. set it to your monitor refresh rate. 60 would mean that we handle a drag event 60 times per second.

Available commands

Click to view the available commands for *Screen*

Bar

class libqtile.bar.**Bar**(**args*, ***kwargs*)

A bar, which can contain widgets

Parameters

widgets
A list of widget objects.

size
The "thickness" of the bar, i.e. the height of a horizontal bar, or the width of a vertical bar.

Configuration options

key	default	description
background	'#000000'	Background colour.
border_color	'#000000'	Border colour as str or list of str [N E S W]
border_width	0	Width of border as int of list of ints [N E S W]
margin	0	Space around bar as int or list of ints [N E S W].
opacity	1	Bar window opacity.
reserve	True	Reserve screen space (when set to 'False', bar will be drawn above windows).

Available commands

Click to view the available commands for *Bar*

Gap

`class libqtile.bar.Gap(size: int)`

A gap placed along one of the edges of the screen

Qtile will avoid covering gaps with windows.

Parameters

size
The "thickness" of the gap, i.e. the height of a horizontal gap, or the width of a vertical gap.

6.3.7 Hooks

Qtile provides a mechanism for subscribing to certain events in `libqtile.hook`. To subscribe to a hook in your configuration, simply decorate a function with the hook you wish to subscribe to.

See *Built-in Hooks* for a listing of available hooks.

Examples

Automatic floating dialogs

Let's say we wanted to automatically float all dialog windows (this code is not actually necessary; Qtile floats all dialogs by default). We would subscribe to the `client_new` hook to tell us when a new window has opened and, if the type is "dialog", as can set the window to float. In our configuration file it would look something like this:

```
from libqtile import hook

@hook.subscribe.client_new
def floating_dialogs(window):
    dialog = window.window.get_wm_type() == 'dialog'
    transient = window.window.get_wm_transient_for()
    if dialog or transient:
        window.floating = True
```

A list of available hooks can be found in the *Built-in Hooks* reference.

Autostart

If you want to run commands or spawn some applications when Qtile starts, you'll want to look at the `startup` and `startup_once` hooks. `startup` is emitted every time Qtile starts (including restarts), whereas `startup_once` is only emitted on the very first startup.

Let's create an executable file `~/.config/qtile/autostart.sh` that will start a few programs when Qtile first runs. Remember to *chmod +x ~/.config/qtile/autostart.sh* so that it can be executed.

```
#!/bin/sh
pidgin &
dropbox start &
```

We can then subscribe to `startup_once` to run this script:

```
import os
import subprocess

from libqtile import hook

@hook.subscribe.startup_once
def autostart():
    home = os.path.expanduser('~/.config/qtile/autostart.sh')
    subprocess.Popen([home])
```

Accessing the qtile object

If you want to do something with the `Qtile` manager instance inside a hook, it can be imported into your config:

```
from libqtile import qtile
```

Async hooks

Hooks can also be defined as coroutine functions using `async def`, which will run them asynchronously in the event loop:

```
@hook.subscribe.focus_change
async def _():
    ...
```

In addition to the above variables, there are several other boolean configuration variables that control specific aspects of Qtile's behavior:

variable	default	description
auto_ful	True	If a window requests to be fullscreen, it is automatically fullscreened. Set this to false if you only want windows to be fullscreen if you ask them to be.
bring_fr	False	When clicked, should the window be brought to the front or not. If this is set to "floating_only", only floating windows will get affected (This sets the X Stack Mode to Above.). This will ignore the layering rules and will therefore bring windows above other windows, even if they have been set as "kept_above". This may cause issues with docks and other similar apps as these may end up hidden behind other windows. Setting this to `False` or `"floating_only"` may therefore be required when using these apps.
cursor_w	False	If true, the cursor follows the focus as directed by the keyboard, warping to the center of the focused window. When switching focus between screens, If there are no windows in the screen, the cursor will warp to the center of the screen.
dgroups_	None	A function which generates group binding hotkeys. It takes a single argument, the DGroups object, and can use that to set up dynamic key bindings. A sample implementation is available in libqtile/dgroups.py called simple_key_binder(), which will bind groups to mod+shift+0-10 by default.
dgroups_	[]	A list of Rule objects which can send windows to various groups based on matching criteria.
extensic	same as widget_d	Default settings for extensions.
floating	layout. Floating ..])	The default floating layout to use. This allows you to set custom floating rules among other things if you wish. See the configuration file for the default *float_rules*.
floats_k	True	Floating windows are kept above tiled windows (Currently x11 only. Wayland support coming soon.)
focus_on	'smart'	Behavior of the _NET_ACTIVATE_WINDOW message sent by applications • urgent: urgent flag is set for the window • focus: automatically focus the window • smart: automatically focus if the window is in the current group • never: never automatically focus any window that requests it
follow_m	True	Controls whether or not focus follows the mouse around as it moves across windows in a layout.
widget_d	dict(fon fontsize padding=	Default settings for bar widgets.
reconfig	True	Controls whether or not to automatically reconfigure screens when there are changes in randr output configuration.
wmname	'LG3D'	Gasp! We're lying here. In fact, nobody really uses or cares about this string besides java UI toolkits; you can see several discussions on the mailing lists, GitHub issues, and other WM documentation that suggest setting this string if your java app doesn't work correctly. We may as well just lie and say that we're a working one by default. We choose LG3D to maximize irony: it is a 3D non-reparenting WM written in java that happens to be on java's whitelist.
auto_min	True	If things like steam games want to auto-minimize themselves when losing focus, should we respect this or not?

6.4 Testing your configuration

The best way to test changes to your configuration is with the provided scripts at ./scripts/xephyr (X11) or ./scripts/wephyr (Wayland). This will run Qtile with your `config.py` inside a nested window and prevent your running instance of Qtile from crashing if something goes wrong.

See *Hacking Qtile* for more information on using Xephyr.

CHAPTER

SEVEN

BUILT-IN LAYOUTS

7.1 Bsp

class libqtile.layout.**Bsp**(**args*, ***kwargs*)

This layout is inspired by bspwm, but it does not try to copy its features.

The first client occupies the entire screen space. When a new client is created, the selected space is partitioned in 2 and the new client occupies one of those subspaces, leaving the old client with the other.

The partition can be either horizontal or vertical according to the dimensions of the current space: if its width/height ratio is above a pre-configured value, the subspaces are created side-by-side, otherwise, they are created on top of each other. The partition direction can be freely toggled. All subspaces can be resized and clients can be shuffled around.

All clients are organized at the leaves of a full binary tree.

An example key configuration is:

```
Key([mod], "j", lazy.layout.down()),
Key([mod], "k", lazy.layout.up()),
Key([mod], "h", lazy.layout.left()),
Key([mod], "l", lazy.layout.right()),
Key([mod, "shift"], "j", lazy.layout.shuffle_down()),
Key([mod, "shift"], "k", lazy.layout.shuffle_up()),
Key([mod, "shift"], "h", lazy.layout.shuffle_left()),
Key([mod, "shift"], "l", lazy.layout.shuffle_right()),
Key([mod, "mod1"], "j", lazy.layout.flip_down()),
Key([mod, "mod1"], "k", lazy.layout.flip_up()),
Key([mod, "mod1"], "h", lazy.layout.flip_left()),
Key([mod, "mod1"], "l", lazy.layout.flip_right()),
Key([mod, "control"], "j", lazy.layout.grow_down()),
Key([mod, "control"], "k", lazy.layout.grow_up()),
Key([mod, "control"], "h", lazy.layout.grow_left()),
Key([mod, "control"], "l", lazy.layout.grow_right()),
Key([mod, "shift"], "n", lazy.layout.normalize()),
Key([mod], "Return", lazy.layout.toggle_split()),
```

Configuration options

key	default	description
border_focus	'#881111'	Border colour(s) for the focused window.
border_normal	'#220000'	Border colour(s) for un-focused windows.
border_on_single	False	Draw border when there is only one window.
border_width	2	Border width.
fair	True	New clients are inserted in the shortest branch.
grow_amount	10	Amount by which to grow a window/column.
lower_right	True	New client occupies lower or right subspace.
margin	0	Margin of the layout (int or list of ints [N E S W]).
margin_on_single	None	Margin when there is only one window (int or list of ints [N E S W], 'None' to use 'margin' value).
ratio	1.6	Width/height ratio that defines the partition direction.
wrap_clients	False	Whether client list should be wrapped when using `next` and `previous` commands.

Available commands

Click to view the available commands for *Bsp*

7.2 Columns

class libqtile.layout.**Columns**(**args*, ***kwargs*)

Extension of the Stack layout.

The screen is split into columns, which can be dynamically added or removed. Each column can present its windows in 2 modes: split or stacked. In split mode, all windows are presented simultaneously, spliting the column space. In stacked mode, only a single window is presented from the stack of windows. Columns and windows can be resized and windows can be shuffled around.

This layout can also emulate wmii's default layout via:

    layout.Columns(num_columns=1, insert_position=1)

Or the "Vertical", and "Max", depending on the default parameters.

An example key configuration is:

```
Key([mod], "j", lazy.layout.down()),
Key([mod], "k", lazy.layout.up()),
Key([mod], "h", lazy.layout.left()),
Key([mod], "l", lazy.layout.right()),
Key([mod, "shift"], "j", lazy.layout.shuffle_down()),
Key([mod, "shift"], "k", lazy.layout.shuffle_up()),
Key([mod, "shift"], "h", lazy.layout.shuffle_left()),
Key([mod, "shift"], "l", lazy.layout.shuffle_right()),
Key([mod, "control"], "j", lazy.layout.grow_down()),
Key([mod, "control"], "k", lazy.layout.grow_up()),
Key([mod, "control"], "h", lazy.layout.grow_left()),
Key([mod, "control"], "l", lazy.layout.grow_right()),
Key([mod, "shift", "control"], "h", lazy.layout.swap_column_left()),
Key([mod, "shift", "control"], "l", lazy.layout.swap_column_right()),
Key([mod], "Return", lazy.layout.toggle_split()),
Key([mod], "n", lazy.layout.normalize()),
```

Configuration options

key	default	description
align	1	Which side of screen new windows will be added to (one of Columns._left or Columns._right). Ignored if 'fair=True'.
border_focus	'#881111'	Border colour(s) for the focused window.
border_focus_sta	'#881111'	Border colour(s) for the focused window in stacked columns.
border_normal	'#220000'	Border colour(s) for un-focused windows.
border_normal_st	'#220000'	Border colour(s) for un-focused windows in stacked columns.
border_on_single	False	Draw a border when there is one only window.
border_width	2	Border width.
fair	False	Add new windows to the column with least windows.
grow_amount	10	Amount by which to grow a window/column.
insert_position	0	Position relative to the current window where new ones are inserted (0 means right above the current window, 1 means right after).
margin	0	Margin of the layout (int or list of ints [N E S W]).
margin_on_single	None	Margin when only one window. (int or list of ints [N E S W])
num_columns	2	Preferred number of columns.
single_border_wi	None	Border width for single window.
split	True	New columns presentation mode.
wrap_focus_colum	True	Wrap the screen when moving focus across columns.
wrap_focus_rows	True	Wrap the screen when moving focus across rows.
wrap_focus_stack	True	Wrap the screen when moving focus across stacked.

Available commands

Click to view the available commands for *Columns*

7.3 Floating

class libqtile.layout.**Floating**(**args*, ***kwargs*)

Floating layout, which does nothing with windows but handles focus order

Configuration options

key	default	description
border_focus	'#0000ff'	Border colour(s) for the focused window.
border_normal	'#000000'	Border colour(s) for un-focused windows.
border_width	1	Border width.
fullscreen_borde	0	Border width for fullscreen.
max_border_width	0	Border width for maximize.

Available commands

Click to view the available commands for *Floating*

7.4 Matrix

class libqtile.layout.**Matrix**(**args*, ***kwargs*)

This layout divides the screen into a matrix of equally sized cells and places one window in each cell. The number of columns is configurable and can also be changed interactively.

Configuration options

key	default	description
border_focus	'#0000ff'	Border colour(s) for the focused window.
border_normal	'#000000'	Border colour(s) for un-focused windows.
border_width	1	Border width.
columns	2	Number of columns
margin	0	Margin of the layout (int or list of ints [N E S W])

Available commands

Click to view the available commands for *Matrix*

7.5 Max

class libqtile.layout.**Max**(**args*, ***kwargs*)

Maximized layout

A simple layout that only displays one window at a time, filling the screen_rect. This is suitable for use on laptops and other devices with small screens. Conceptually, the windows are managed as a stack, with commands to switch to next and previous windows in the stack.

Configuration options

key	default	description
border_focus	'#0000ff'	Border colour(s) for the window when focused
border_normal	'#000000'	Border colour(s) for the window when not focused
border_width	0	Border width.
margin	0	Margin of the layout (int or list of ints [N E S W])
only_focused	True	Only draw the focused window

Available commands

Click to view the available commands for *Max*

7.6 MonadTall

class libqtile.layout.**MonadTall**(**args*, ***kwargs*)

Emulate the behavior of XMonad's default tiling scheme.

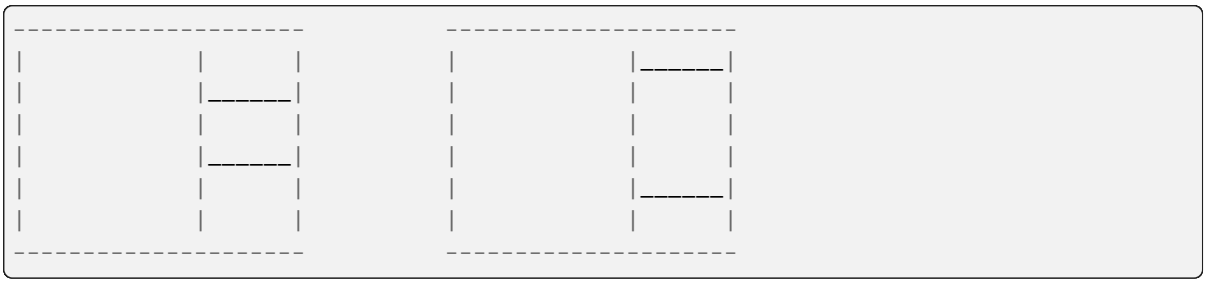
Main-Pane:

A main pane that contains a single window takes up a vertical portion of the screen_rect based on the ratio setting. This ratio can be adjusted with the `grow_main` and `shrink_main` or, while the main pane is in focus, `grow` and `shrink`. You may also set the ratio directly with `set_ratio`.

Using the `flip` method will switch which horizontal side the main pane will occupy. The main pane is considered the "top" of the stack.

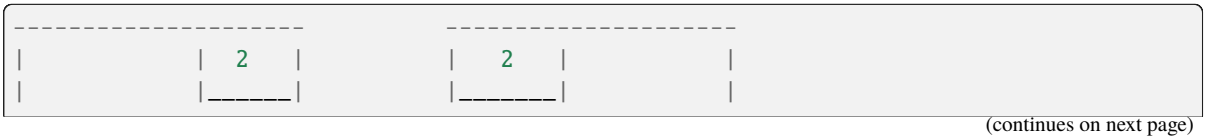
Secondary-panes:

Occupying the rest of the screen_rect are one or more secondary panes. The secondary panes will share the vertical space of the screen_rect however they can be resized at will with the `grow` and `shrink` methods. The other secondary panes will adjust their sizes to smoothly fill all of the space.



Panes can be moved with the `shuffle_up` and `shuffle_down` methods. As mentioned the main pane is considered the top of the stack; moving up is counter-clockwise and moving down is clockwise.

The opposite is true if the layout is "flipped".



(continues on next page)

(continued from previous page)

```
|             |  3   |          |   3   |           |
|     1       |______|          |_______|     1     |
|             |  4   |          |   4   |           |
|             |      |          |       |           |
----------------------          ---------------------
```

Normalizing/Resetting:

To restore all secondary client windows to their default size ratios use the `normalize` method.

To reset all client windows to their default sizes, including the primary window, use the `reset` method.

Maximizing:

To toggle a client window between its minimum and maximum sizes simply use the `maximize` on a focused client.

Suggested Bindings:

```
Key([modkey], "h", lazy.layout.left()),
Key([modkey], "l", lazy.layout.right()),
Key([modkey], "j", lazy.layout.down()),
Key([modkey], "k", lazy.layout.up()),
Key([modkey, "shift"], "h", lazy.layout.swap_left()),
Key([modkey, "shift"], "l", lazy.layout.swap_right()),
Key([modkey, "shift"], "j", lazy.layout.shuffle_down()),
Key([modkey, "shift"], "k", lazy.layout.shuffle_up()),
Key([modkey], "i", lazy.layout.grow()),
Key([modkey], "m", lazy.layout.shrink()),
Key([modkey], "n", lazy.layout.reset()),
Key([modkey, "shift"], "n", lazy.layout.normalize()),
Key([modkey], "o", lazy.layout.maximize()),
Key([modkey, "shift"], "space", lazy.layout.flip()),
```

Configuration options

key	default	description
align	0	Which side master plane will be placed (one of MonadTall._left or MonadTall._right)
border_focus	'#ff0000'	Border colour(s) for the focused window.
border_normal	'#000000'	Border colour(s) for un-focused windows.
border_width	2	Border width.
change_ratio	0.05	Resize ratio
change_size	20	Resize change in pixels
margin	0	Margin of the layout
max_ratio	0.75	The percent of the screen-space the master pane should occupy at maximum.
min_ratio	0.25	The percent of the screen-space the master pane should occupy at minimum.
min_secondary_si	85	minimum size in pixel for a secondary pane window
new_client_posit	'after_current'	Place new windows: after_current - after the active window. before_current - before the active window, top - at the top of the stack, bottom - at the bottom of the stack,
ratio	0.5	The percent of the screen-space the master pane should occupy by default.
single_border_wi	None	Border width for single window
single_margin	None	Margin size for single window

Available commands

Click to view the available commands for *MonadTall*

7.7 MonadThreeCol

class libqtile.layout.**MonadThreeCol**(**args*, ***kwargs*)

Emulate the behavior of XMonad's ThreeColumns layout.

A layout similar to tall but with three columns. With an ultra wide display this layout can be used for a huge main window - ideally at the center of the screen - and up to six reasonable sized secondary windows.

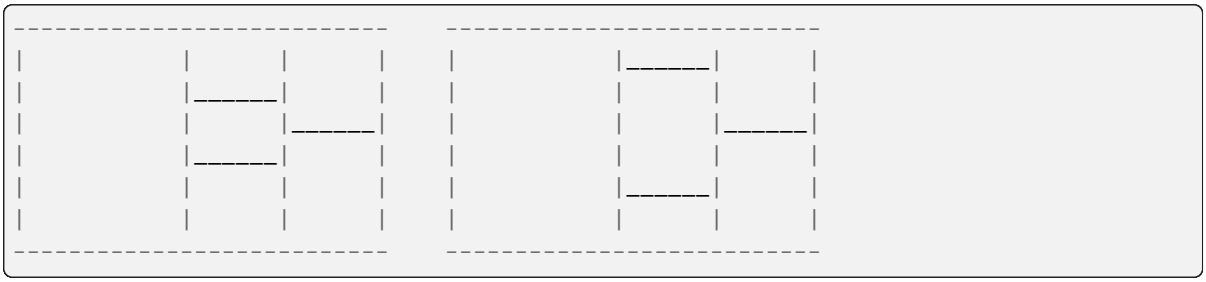
Main-Pane:

A main pane that contains a single window takes up a vertical portion of the screen_rect based on the ratio setting. This ratio can be adjusted with the grow_main and shrink_main or, while the main pane is in focus, grow and shrink. The main pane can also be centered.

Secondary-panes:

Occupying the rest of the screen_rect are one or more secondary panes. The secondary panes will be divided into two columns and share the vertical space of each column. However they can be resized at will with the grow

and `shrink` methods. The other secondary panes will adjust their sizes to smoothly fill all of the space.



Panes can be moved with the `shuffle_up` and `shuffle_down` methods. As mentioned the main pane is considered the top of the stack; moving up is counter-clockwise and moving down is clockwise. A secondary pane can also be promoted to the main pane with the `swap_main` method.

Normalizing/Resetting:

To restore all secondary client windows to their default size ratios use the `normalize` method.

To reset all client windows to their default sizes, including the primary window, use the `reset` method.

Maximizing:

To maximized a client window simply use the `maximize` on a focused client.

Configuration options

key	default	description
`align`	`0`	Which side master plane will be placed (one of `MonadTall._left` or `MonadTall._right`)
`border_focus`	`'#ff0000'`	Border colour(s) for the focused window.
`border_normal`	`'#000000'`	Border colour(s) for un-focused windows.
`border_width`	`2`	Border width.
`change_ratio`	`0.05`	Resize ratio
`change_size`	`20`	Resize change in pixels
`main_centered`	`True`	Place the main pane at the center of the screen
`margin`	`0`	Margin of the layout
`max_ratio`	`0.75`	The percent of the screen-space the master pane should occupy at maximum.
`min_ratio`	`0.25`	The percent of the screen-space the master pane should occupy at minimum.
`min_secondary_si`	`85`	minimum size in pixel for a secondary pane window
`new_client_posit`	`'top'`	Place new windows: after_current - after the active window. before_current - before the active window, top - at the top of the stack, bottom - at the bottom of the stack,
`ratio`	`0.5`	The percent of the screen-space the master pane should occupy by default.
`single_border_wi`	`None`	Border width for single window
`single_margin`	`None`	Margin size for single window

Available commands

Click to view the available commands for *MonadThreeCol*

7.8 MonadWide

class libqtile.layout.**MonadWide**(**args*, ***kwargs*)

Emulate the behavior of XMonad's horizontal tiling scheme.

This layout attempts to emulate the behavior of XMonad wide tiling scheme.

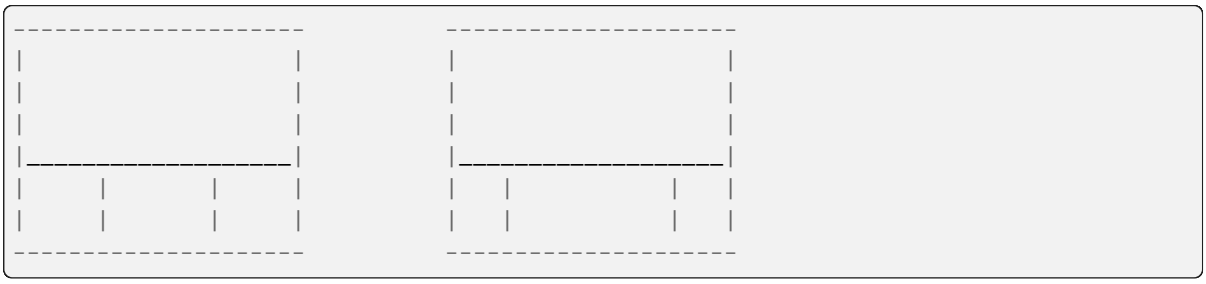
Main-Pane:

A main pane that contains a single window takes up a horizontal portion of the screen_rect based on the ratio setting. This ratio can be adjusted with the `grow_main` and `shrink_main` or, while the main pane is in focus, `grow` and `shrink`.

Using the `flip` method will switch which vertical side the main pane will occupy. The main pane is considered the "top" of the stack.

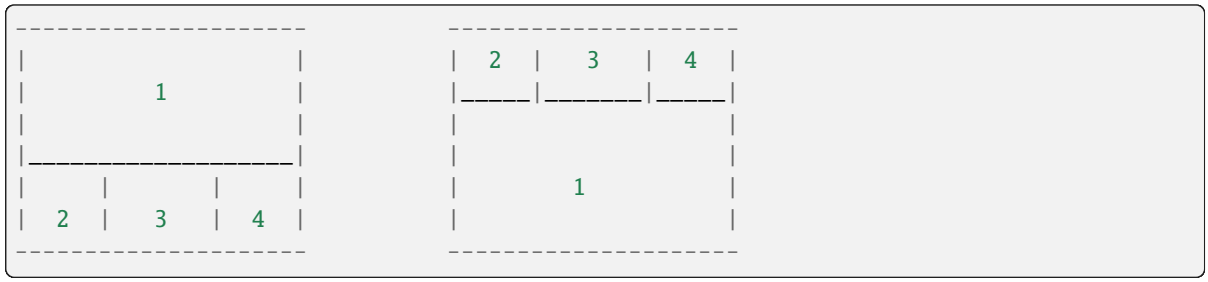
Secondary-panes:

Occupying the rest of the screen_rect are one or more secondary panes. The secondary panes will share the horizontal space of the screen_rect however they can be resized at will with the `grow` and `shrink` methods. The other secondary panes will adjust their sizes to smoothly fill all of the space.



Panes can be moved with the `shuffle_up` and `shuffle_down` methods. As mentioned the main pane is considered the top of the stack; moving up is counter-clockwise and moving down is clockwise.

The opposite is true if the layout is "flipped".



Normalizing/Resetting:

To restore all secondary client windows to their default size ratios use the `normalize` method.

To reset all client windows to their default sizes, including the primary window, use the `reset` method.

Maximizing:

To toggle a client window between its minimum and maximum sizes simply use the `maximize` on a focused client.

Suggested Bindings:

```
Key([modkey], "h", lazy.layout.left()),
Key([modkey], "l", lazy.layout.right()),
Key([modkey], "j", lazy.layout.down()),
Key([modkey], "k", lazy.layout.up()),
Key([modkey, "shift"], "h", lazy.layout.swap_left()),
Key([modkey, "shift"], "l", lazy.layout.swap_right()),
Key([modkey, "shift"], "j", lazy.layout.shuffle_down()),
Key([modkey, "shift"], "k", lazy.layout.shuffle_up()),
Key([modkey], "i", lazy.layout.grow()),
Key([modkey], "m", lazy.layout.shrink()),
Key([modkey], "n", lazy.layout.reset()),
Key([modkey, "shift"], "n", lazy.layout.normalize()),
Key([modkey], "o", lazy.layout.maximize()),
Key([modkey, "shift"], "space", lazy.layout.flip()),
```

Configuration options

key	default	description
align	0	Which side master plane will be placed (one of MonadTall._left or MonadTall._right)
border_focus	'#ff0000'	Border colour(s) for the focused window.
border_normal	'#000000'	Border colour(s) for un-focused windows.
border_width	2	Border width.
change_ratio	0.05	Resize ratio
change_size	20	Resize change in pixels
margin	0	Margin of the layout
max_ratio	0.75	The percent of the screen-space the master pane should occupy at maximum.
min_ratio	0.25	The percent of the screen-space the master pane should occupy at minimum.
min_secondary_si	85	minimum size in pixel for a secondary pane window
new_client_posit	'after_current'	Place new windows: after_current - after the active window. before_current - before the active window, top - at the top of the stack, bottom - at the bottom of the stack,
ratio	0.5	The percent of the screen-space the master pane should occupy by default.
single_border_wi	None	Border width for single window
single_margin	None	Margin size for single window

Available commands

Click to view the available commands for *MonadWide*

7.9 RatioTile

class libqtile.layout.**RatioTile**(**args*, ***kwargs*)

Tries to tile all windows in the width/height ratio passed in

Configuration options

key	default	description
border_focus	'#0000ff'	Border colour(s) for the focused window.
border_normal	'#000000'	Border colour(s) for un-focused windows.
border_width	1	Border width.
fancy	False	Use a different method to calculate window sizes.
margin	0	Margin of the layout (int or list of ints [N E S W])
ratio	1.618	Ratio of the tiles
ratio_increment	0.1	Amount to increment per ratio increment

Available commands

Click to view the available commands for *RatioTile*

7.10 ScreenSplit

class libqtile.layout.**ScreenSplit**(**args*, ***kwargs*)

A layout that allows you to split the screen into separate areas, each of which can be assigned its own layout.

This layout is intended to be used on large monitors where separate layouts may be desirable. However, unlike creating virtual screens, this layout retains the full screen configuration meaning that full screen windows will continue to fill the entire screen.

Each split is defined as a dictionary with the following keys:

- `name`: this is used with the `ScreenSplit` widget (see below)
- `rect`: a tuple of (x, y, width, height) with each value being between 0 and 1. These are relative values based on the screen's dimensions e.g. a value of `(0.5, 0, 0.5, 1)` would define an area starting at the top middle of the screen and extending to the bottom left corner.
- `layout`: the layout to occupy the defined split.
- `matches`: (optional) list of `Match` objects which define which windows will open in the defined split.

Different splits can be selected by using the following `lazy.layout.next_split()` and `lazy.layout.previous_split()` commands.

To identify which split is active, users can use the `ScreenSplit` widget will show the name of the split and the relevant layout. Scrolling up and down on the widget will change the active split.

Note: While keybindings will be passed to the active split's layout, bindings using the `.when(layout=...)`` syntax will not be applied as the primary layout is `ScreenSplit`.

Configuration options

key	default	description
splits	[{'layout': <libqtile.layout.max.Max object at 0x7fa5f9739840>, 'name': 'top', 'rect': (0, 0, 1, 0.5)}, {'layout': <libqtile.layout.columns.Columns object at 0x7fa5f9739c00>, 'name': 'bottom', 'rect': (0, 0.5, 1, 0.5)}]	Screen splits. See documentation for details.

Available commands

Click to view the available commands for *ScreenSplit*

7.11 Slice

class libqtile.layout.**Slice**(**args*, ***kwargs*)

Slice layout

This layout cuts piece of screen_rect and places a single window on that piece, and delegates other window placement to other layout

Configuration options

key	default	description
`fallback`	`<libqtile.layout.max.Max object at 0x7fa5f97a0d60>`	Layout to be used for the non-slice area.
`match`	`None`	Match-object describing which window(s) to move to the slice.
`side`	`'left'`	Position of the slice (left, right, top, bottom).
`width`	`256`	Slice width.

Available commands

Click to view the available commands for *Slice*

7.12 Spiral

class libqtile.layout.**Spiral**(**args*, ***kwargs*)

A mathematical layout.

Renders windows in a spiral form by splitting the screen based on a selected ratio. The direction of the split is changed every time in a defined order resulting in a spiral formation.

The main window can be sized with `lazy.layout.grow_main()` and `lazy.layout.shrink_main()`. All other windows are sized by `lazy.layout.increase_ratio()` and `lazy.layout.decrease_ratio()`.

NB if `main_pane_ratio` is not set then it will also be adjusted according to `ratio`. However, as soon `shrink_main()` or `grow_main()` have been called once then the master pane will only change size following further calls to those methods.

Users are able to choose the location of the main (i.e. largest) pane and the direction of the rotation.

Some examples:

`main_pane="left", clockwise=True`

```
----------------------
|1        |2         |
|         |          |
|         |          |
|         |----------|
|         |5 |6 |3   |
```

(continues on next page)

(continued from previous page)

```
|         |-----|    |
|         |4    |    |
----------------------
```

`main_pane="top", clockwise=False`

```
----------------------
|1                   |
|                    |
|                    |
|--------------------|
|2         |5    |4  |
|          |-----------|
|          |3          |
----------------------
```

Configuration options

key	default	description
`border_focus`	`'#0000ff'`	Border colour(s) for the focused window.
`border_normal`	`'#000000'`	Border colour(s) for un-focused windows.
`border_width`	`1`	Border width.
`clockwise`	`True`	Direction of spiral
`main_pane`	`'left'`	Location of biggest window 'top', 'bottom', 'left', 'right'
`main_pane_ratio`	`None`	Ratio for biggest window or 'None' to use same ratio for all windows.
`margin`	`0`	Margin of the layout (int or list of ints [N E S W])
`new_client_posit`	`'top'`	Place new windows: 'after_current' - after the active window, 'before_current' - before the active window, 'top' - in the main pane, 'bottom '- at the bottom of the stack. NB windows that are added too low in the stack may be hidden if there is no remaining space in the spiral.
`ratio`	`0.6180469715698392`	Ratio of the tiles
`ratio_increment`	`0.1`	Amount to increment per ratio increment

Available commands

Click to view the available commands for *Spiral*

7.13 Stack

class libqtile.layout.**Stack**(**args*, ***kwargs*)

A layout composed of stacks of windows

The stack layout divides the screen_rect horizontally into a set of stacks. Commands allow you to switch between stacks, to next and previous windows within a stack, and to split a stack to show all windows in the stack, or unsplit it to show only the current window.

Unlike the columns layout the number of stacks is fixed.

Configuration options

key	default	description
autosplit	False	Auto split all new stacks.
border_focus	'#0000ff'	Border colour(s) for the focused window.
border_focus_sta	None	Border colour(s) for the focused stacked window. If 'None' will default to border_focus.
border_normal	'#000000'	Border colour(s) for un-focused windows.
border_normal_st	None	Border colour(s) for un-focused stacked windows. If 'None' will default to border_normal.
border_width	1	Border width.
fair	False	Add new windows to the stacks in a round robin way.
margin	0	Margin of the layout (int or list of ints [N E S W])
num_stacks	2	Number of stacks.

Available commands

Click to view the available commands for *Stack*

7.14 Tile

class libqtile.layout.**Tile**(**args*, ***kwargs*)

A layout with two stacks of windows dividing the screen

The Tile layout divides the screen_rect horizontally into two stacks. The maximum amount of "master" windows can be configured; surplus windows will be displayed in the slave stack on the right. Within their stacks, the windows will be tiled vertically. The windows can be rotated in their entirety by calling up() or down() or, if shift_windows is set to True, individually.

Configuration options

key	default	description
add_after_last	False	Add new clients after all the others. If this is True, it overrides add_on_top.
add_on_top	True	Add new clients before all the others, potentially pushing other windows into slave stack.
border_focus	'#0000ff'	Border colour(s) for the focused window.
border_normal	'#000000'	Border colour(s) for un-focused windows.
border_on_single	True	Whether to draw border if there is only one window.
border_width	1	Border width.
expand	True	Expand the master windows to the full screen width if no slaves are present.
margin	0	Margin of the layout (int or list of ints [N E S W])
margin_on_single	True	Whether to draw margin if there is only one window.
master_length	1	Amount of windows displayed in the master stack. Surplus windows will be moved to the slave stack.
master_match	None	A Match object defining which window(s) should be kept masters (single or a list of Match-objects).
max_ratio	0.85	Maximum width of master windows
min_ratio	0.15	Minimum width of master windows
ratio	0.618	Width-percentage of screen size reserved for master windows.
ratio_increment	0.05	By which amount to change ratio when decrease_ratio or increase_ratio are called.
shift_windows	False	Allow to shift windows within the layout. If False, the layout will be rotated instead.

Available commands

Click to view the available commands for *Tile*

7.15 TreeTab

class libqtile.layout.**TreeTab**(**args*, ***kwargs*)

Tree Tab Layout

This layout works just like Max but displays tree of the windows at the left border of the screen_rect, which allows you to overview all opened windows. It's designed to work with `uzbl-browser` but works with other windows too.

The panel at the left border contains sections, each of which contains windows. Initially the panel looks like flat lists inside its section, and looks like trees if some of the windows are "moved" left or right.

For example, it looks like below with two sections initially:

```
+------------+
|Section Foo |
+------------+
| Window A   |
+------------+
| Window B   |
+------------+
| Window C   |
```

(continues on next page)

(continued from previous page)

```
+------------+
|Section Bar |
+------------+
```

And then it will look like below if "Window B" is moved right and "Window C" is moved right too:

```
+------------+
|Section Foo |
+------------+
| Window A   |
+------------+
|  Window B  |
+------------+
|   Window C |
+------------+
|Section Bar |
+------------+
```

Configuration options

key	default	description
active_bg	'000080'	Background color of active tab
active_fg	'ffffff'	Foreground color of active tab
bg_color	'000000'	Background color of tabs
border_width	2	Width of the border
font	'sans'	Font
fontshadow	None	font shadow color, default is None (no shadow)
fontsize	14	Font pixel size.
inactive_bg	'606060'	Background color of inactive tab
inactive_fg	'ffffff'	Foreground color of inactive tab
level_shift	8	Shift for children tabs
margin_left	6	Left margin of tab panel
margin_y	6	Vertical margin of tab panel
padding_left	6	Left padding for tabs
padding_x	6	Left padding for tab label
padding_y	2	Top padding for tab label
panel_width	150	Width of the left panel
place_right	False	Place the tab panel on the right side
previous_on_rm	False	Focus previous window on close instead of first.
section_bottom	6	Bottom margin of section
section_fg	'ffffff'	Color of section label
section_fontsize	11	Font pixel size of section label
section_left	4	Left margin of section label
section_padding	4	Bottom of margin section label
section_top	4	Top margin of section label
sections	['Default']	Titles of section instances
urgent_bg	'ff0000'	Background color of urgent tab
urgent_fg	'ffffff'	Foreground color of urgent tab
vspace	2	Space between tabs

Available commands

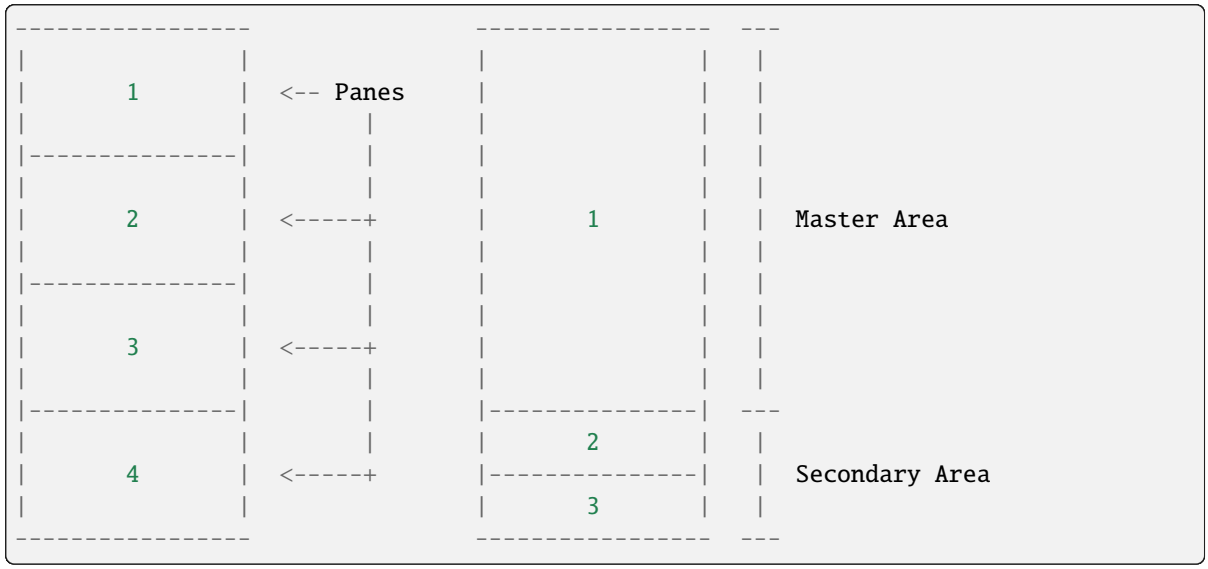
Click to view the available commands for *TreeTab*

7.16 VerticalTile

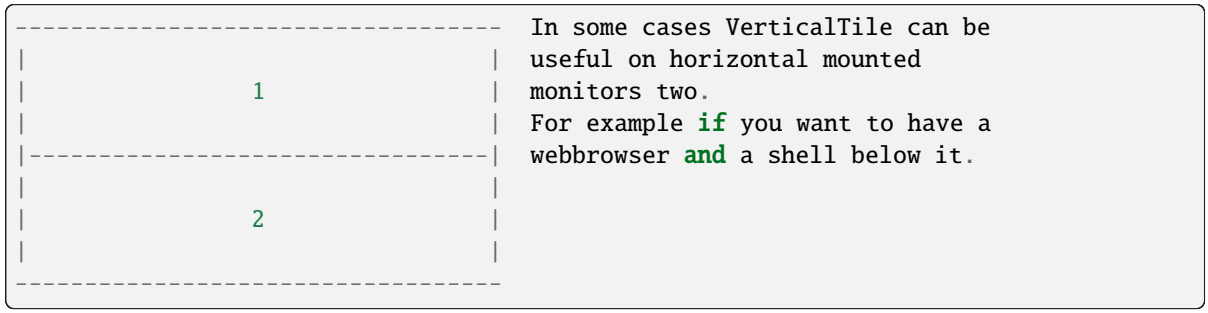
class libqtile.layout.**VerticalTile**(**args*, ***kwargs*)

Tiling layout that works nice on vertically mounted monitors

The available height gets divided by the number of panes, if no pane is maximized. If one pane has been maximized, the available height gets split in master- and secondary area. The maximized pane (master pane) gets the full height of the master area and the other panes (secondary panes) share the remaining space. The master area (at default 75%) can grow and shrink via keybindings.



Normal behavior. No One maximized pane in the master area maximized pane. No and two secondary panes in the specific areas. secondary area.



Suggested keybindings:

```
Key([modkey], 'j', lazy.layout.down()),
Key([modkey], 'k', lazy.layout.up()),
Key([modkey], 'Tab', lazy.layout.next()),
Key([modkey, 'shift'], 'Tab', lazy.layout.next()),
Key([modkey, 'shift'], 'j', lazy.layout.shuffle_down()),
Key([modkey, 'shift'], 'k', lazy.layout.shuffle_up()),
Key([modkey], 'm', lazy.layout.maximize()),
Key([modkey], 'n', lazy.layout.normalize()),
```

Configuration options

key	default	description
`border_focus`	`'#FF0000'`	Border color(s) for the focused window.
`border_normal`	`'#FFFFFF'`	Border color(s) for un-focused windows.
`border_width`	`1`	Border width.
`margin`	`0`	Border margin (int or list of ints [N E S W]).
`single_border_wi`	`None`	Border width for single window.
`single_margin`	`None`	Margin for single window.

Available commands

Click to view the available commands for *VerticalTile*

7.17 Zoomy

class libqtile.layout.**Zoomy**(**args*, ***kwargs*)

A layout with single active windows, and few other previews at the right

Configuration options

key	default	description
`columnwidth`	`150`	Width of the right column
`margin`	`0`	Margin of the layout (int or list of ints [N E S W])
`property_big`	`'1.0'`	Property value to set on normal window (X11 only)
`property_name`	`'ZOOM'`	Property to set on zoomed window (X11 only)
`property_small`	`'0.1'`	Property value to set on zoomed window (X11 only)

Available commands

Click to view the available commands for *Zoomy*

CHAPTER

EIGHT

BUILT-IN WIDGETS

8.1 AGroupBox

class libqtile.widget.**AGroupBox**(**args*, ***kwargs*)

A widget that graphically displays the current group

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
border	'000000'	group box border color
borderwidth	3	Current group border width
center_aligned	True	center-aligned group box
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
markup	False	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *AGroupBox*

8.2 Backlight

class libqtile.widget.**Backlight**(**args*, ***kwargs*)

A simple widget to show the current brightness of a monitor.

If the change_command parameter is set to None, the widget will attempt to use the interface at /sys/class to change brightness. This depends on having the correct udev rules, so be sure Qtile's udev rules are installed correctly.

You can also bind keyboard shortcuts to the backlight widget with:

```
from libqtile.widget import backlight
Key(
    [],
    "XF86MonBrightnessUp",
    lazy.widget['backlight'].change_backlight(backlight.ChangeDirection.UP)
)
Key(
    [],
    "XF86MonBrightnessDown",
    lazy.widget['backlight'].change_backlight(backlight.ChangeDirection.DOWN)
)
```

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
backlight_name	'acpi_video0'	ACPI name of a backlight device
brightness_file	'brightness'	Name of file with the current brightness in /sys/class/backlight/backlight_name
change_command	'xbacklight -set {0}'	Execute command to change value
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{percent:2.0%}'	Display format
markup	True	Whether or not to use pango markup
max_brightness_f	'max_brightness'	Name of file with the maximum brightness in /sys/class/backlight/backlight_name
max_chars	0	Maximum number of characters to display in widget.
min_brightness	0	Minimum brightness percentage
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
step	10	Percent of backlight every scroll changed
update_interval	0.2	The delay in seconds between updates

Available commands

Click to view the available commands for *Backlight*

8.3 Battery

class libqtile.widget.**Battery**(**args*, ***kwargs*)

A text-based battery monitoring widget supporting both Linux and FreeBSD.

The Linux version of this widget has functionality to charge "smartly" (i.e. not to 100%) under user defined conditions, and provides some implementations for doing so. For example, to only charge the battery to 90%, use:

```
Battery(..., charge_controller: lambda (0, 90))
```

The battery widget also supplies some charging algorithms. To only charge the battery between 40-50% while connected to a thunderbolt docking station, but 90% all other times, use:

```
from libqtile.widget.battery import thunderbolt_smart_charge
Battery(..., charge_controller: thunderbolt_smart_charge)
```

To temporarily disable/re-enable this (e.g. if you know you're going mobile and need to charge) use either:

```
qtile cmd-obj -o bar top widget battery -f charge_to_full
qtile cmd-obj -o bar top widget battery -f charge_dynamically
```

or bind a key to:

```
Key([mod, "shift"], "c", lazy.widget['battery'].charge_to_full())
Key([mod, "shift"], "x", lazy.widget['battery'].charge_dynamically())
```

note that this functionality requires qtile to be able to write to certain files in sysfs, so make sure that qtile's udev rules are installed correctly.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
battery	0	Which battery should be monitored (battery number or name)
charge_char	'^'	Character to indicate the battery is charging
discharge_char	'V'	Character to indicate the battery is discharging
empty_char	'x'	Character to indicate the battery is empty
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` parameter of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{char} {percent:2.0%} {hour:d}:{min:02 {watt:.2f} W'	Display format

continues on next page

Table 1 – continued from previous page

key	default	description
full_char	'='	Character to indicate the battery is full
hide_threshold	None	Hide the text when there is enough energy 0 <= x < 1
low_background	None	Background color on low battery
low_foreground	'FF0000'	Font color on low battery
low_percentage	0.1	Indicates when to use the low_foreground color 0 < x < 1
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
not_charging_cha	'*'	Character to indicate the batter is not charging
notification_tim	10	Time in seconds to display notification. 0 for no expiry.
notify_below	None	Send a notification below this battery level.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
show_short_text	True	Show "Full" or "Empty" rather than formated text
unknown_char	'?'	Character to indicate the battery status is unknown
update_interval	60	Seconds between status updates

Available commands

Click to view the available commands for *Battery*

8.4 BatteryIcon

class libqtile.widget.**BatteryIcon**(**args*, ***kwargs*)

Battery life indicator widget.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
battery	0	Which battery should be monitored
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	0	Additional padding either side of the icon
scale	1	Scale factor relative to the bar height. Defaults to 1
theme_path	'/home/docs/checkouts/readthedocs.org/user_builds/qtile/checkouts/latest/libqtile/resources/battery-icons'	Path of the icons
update_interval	60	Seconds between status updates

Available commands

Click to view the available commands for *BatteryIcon*

8.5 Bluetooth

class libqtile.widget.**Bluetooth**(**args*, ***kwargs*)

Bluetooth widget that provides following functionality: - View multiple adapters/devices (adapters can be filtered) - Set power and discovery status for adapters - Connect/disconnect/pair devices

The widget works by providing a menu in the bar. Different items are accessed by scrolling up and down on the widget.

Clicking on an adapter will open a submenu allowing you to set power and discovery status.

Clicking on a device will perform an action based on the status of that device: - Connected devices will be disconnected - Disconnected devices will be connected - Unpaired devices (which appear if discovery is on) will be paired and connected

Symbols are used to show the status of adapters and devices.

Battery level for bluetooth devices can also be shown if available. This functionality is not available by default on all distros. If it doesn't work, you can try adding `Experimental = true` to `/etc/bluetooth/main.conf`.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
adapter_format	'Adapter: {name} [{powered}{disco	Text to display when showing adapter device.

continues on next page

Table 2 – continued from previous page

key	default	description
adapter_paths	[]	List of DBus object path for bluetooth adapter (e.g. '/org/bluez/hci0'). Empty list will show all adapters.
background	None	Widget background color
default_show_bat	False	Include battery level of 'connected_devices' in 'default_text'. Uses 'device_battery_format'.
default_text	'BT {connected_devic	Text to show when not scrolling through menu. Available fields: 'connected_devices' list of connected devices, 'num_connected_devices' number of connected devices, 'adapters' list of bluetooth adapters, 'num_adapters' number of bluetooth adapters.
default_timeout	None	Time before reverting to default_text. If 'None', text will stay on selected item.
device	None	Device path, can be found with d-feet or similar dbus explorer. When set, the widget will default to showing this device status.
device_battery_f	' ({battery}%)'	Text to be shown if device reports battery level
device_format	'Device: {name}{battery_l [{symbol}]'	Text to display when showing bluetooth device. The {adapter field is also available if you're using multiple adapters.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
hci	None	(deprecated) same as 'device'.
hide_unnamed_dev	False	Devices with no name will be hidden from scan results
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step

continues on next page

Table 2 – continued from previous page

key	default	description
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator	', '	Separator for lists in 'default_text'.
symbol_connected	'*'	Symbol to indicate device is connected
symbol_discovery	('D', '')	Symbols when adapter is discovering and not discovering
symbol_paired	'-'	Symbol to indicate device is paired but unconnected
symbol_powered	('*', '-')	Symbols when adapter is powered and unpowered.
symbol_unknown	'?'	Symbol to indicate device is unpaired

Available commands

Click to view the available commands for *Bluetooth*

8.6 CPU

class libqtile.widget.**CPU**(**args*, ***kwargs*)

A simple widget to display CPU load and frequency.

Widget requirements: psutil.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'CPU {freq_current}GH {load_percent}%'	CPU display format
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1.0	Update interval for the CPU widget

Available commands

Click to view the available commands for *CPU*

8.7 CPUGraph

class libqtile.widget.**CPUGraph**(**args*, ***kwargs*)

Display CPU usage graph.

Widget requirements: psutil.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
border_color	'215578'	Widget border color
border_width	2	Widget border width
core	'all'	Which core to show (all/0/1/2/...)
fill_color	'1667EB.3'	Fill color for linefill graph
frequency	1	Update frequency in seconds
graph_color	'18BAEB'	Graph color
line_width	3	Line width
margin_x	3	Margin X
margin_y	3	Margin Y
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
samples	100	Count of graph samples.
start_pos	'bottom'	Drawer starting position ('bottom'/'top')
type	'linefill'	'box', 'line', 'linefill'

Available commands

Click to view the available commands for *CPUGraph*

8.8 Canto

class libqtile.widget.**Canto**(**args*, ***kwargs*)

Display RSS feeds updates using the canto console reader

Widget requirements: canto

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
all_format	'{number}'	All feeds display format
background	None	Widget background color
feeds	[]	List of feeds to display, empty for all
fetch	False	Whether to fetch new items on update
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
one_format	'{name}: {number}'	One feed display format
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	600	Update interval in seconds, if none, the widget updates only once.

Available commands

Click to view the available commands for *Canto*

8.9 CapsNumLockIndicator

class libqtile.widget.**CapsNumLockIndicator**(**args*, ***kwargs*)

Really simple widget to show the current Caps/Num Lock state.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	0.5	Update Time in seconds.

Available commands

Click to view the available commands for *CapsNumLockIndicator*

8.10 CheckUpdates

class libqtile.widget.**CheckUpdates**(**args*, ***kwargs*)

Shows number of pending updates in different unix systems.

The following built-in options are available via the distro parameter:

- 'Arch' runs ('pacman -Qu', 0)
- 'Arch_checkupdates' runs ('checkupdates', 0)
- 'Arch_Sup' runs ('pacman -Sup', 0)
- 'Arch_paru' runs ('paru -Qu', 0)
- 'Arch_paru_Sup' runs ('paru -Sup', 0)
- 'Arch_yay' runs ('yay -Qu', 0)

- 'Debian' runs ('apt-show-versions -u -b', 0)
- 'Gentoo_eix' runs ('EIX_LIMIT=0 eix -u# --world', 0)
- 'Guix' runs ('guix upgrade --dry-run', 0)
- 'Ubuntu' runs ('aptitude search ~U', 0)
- 'Fedora' runs ('dnf list updates -q', 1)
- 'FreeBSD' runs ("pkg_version -I -l '<'", 0)
- 'Mandriva' runs ('urpmq --auto-select', 0)
- 'Void' runs ('xbps-install -nuMS', 0)

Note: It is common for package managers to return a non-zero code when there are no updates. As a result, the widget will treat *any* error as if there are no updates. If you are using a custom commmand/script, you should therefore ensure that it returns zero when it completes if you wish to see the output of your command.

In addition, as no errors are recorded to the log, if the widget is showing no updates and you believe that to be incorrect, you should run the appropriate command in a terminal to view any error messages.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
colour_have_upda	'ffffff'	Colour when there are updates.
colour_no_update	'ffffff'	Colour when there's no updates.
custom_command	None	Custom shell command for checking updates (counts the lines of the output)
custom_command_m	<function CheckUpdates. <lambda> at 0x7fa5f8233130>	Lambda function to modify line count from custom_command
display_format	'Updates: {updates}'	Display format if updates available
distro	'Arch'	Name of your distribution
execute	None	Command to execute on click
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
initial_text	''	Draw the widget immediately with an initial text, useful if it takes time to check system updates.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
no_update_string	''	String to display if no updates available
padding	None	Padding. Calculated if None.
restart_indicato	''	Indicator to represent reboot is required. (Ubuntu only)
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	60	Update interval in seconds.

Available commands

Click to view the available commands for *CheckUpdates*

8.11 Chord

class libqtile.widget.**Chord**(**args*, ***kwargs*)

Display current key chord

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
chords_colors	{}	colors per chord in form of tuple {'chord_name': ('bg', 'fg')}. Where a chord name is not in the dictionary, the default background and foreground values will be used.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
name_transform	<function Chord. <lambda> at 0x7fa5f8233490>	preprocessor for chord name it is pure function string -> string
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *Chord*

8.12 Clipboard

class libqtile.widget.**Clipboard**(**args*, ***kwargs*)

Display current clipboard contents

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
blacklist	['keepassx']	list with blacklisted wm_class, sadly not every clipboard window sets them, keepassx does.Clipboard contents from blacklisted wm_classes will be replaced by the value of blacklist_text.
blacklist_text	'***********'	text to display when the wm_class is blacklisted
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
max_width	10	maximum number of characters to display (None for all, useful when width is bar.STRETCH)
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
selection	'CLIPBOARD'	the selection to display(CLIPBOARD or PRIMARY)
timeout	10	Default timeout (seconds) for display text, None to keep forever

Available commands

Click to view the available commands for *Clipboard*

8.13 Clock

class libqtile.widget.**Clock**(**args*, ***kwargs*)

A simple but flexible text-based clock

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` parameter of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'%H:%M'	A Python datetime format string
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
timezone	None	The timezone to use for this clock, either as string if pytz or dateutil is installed (e.g. "US/Central" or anything in /usr/share/zoneinfo), or as tzinfo (e.g. datetime.timezone.utc). None means the system local timezone and is the default.
update_interval	1.0	Update interval for the clock

Available commands

Click to view the available commands for *Clock*

8.14 Cmus

class libqtile.widget.**Cmus**(**args*, ***kwargs*)

A simple Cmus widget.

Show the metadata of now listening song and allow basic mouse control from the bar:

- toggle pause (or play if stopped) on left click;
- skip forward in playlist on scroll up;
- skip backward in playlist on scroll down.

Cmus (https://cmus.github.io) should be installed.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{play_icon}{artist} - {title}'	Format of playback info.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
noplay_color	'cecece'	Text colour when not playing.
padding	None	Padding. Calculated if None.
play_color	'00ff00'	Text colour when playing.
play_icon	' '	Icon to display, if chosen.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	0.5	Update Time in seconds.

Available commands

Click to view the available commands for *Cmus*

8.15 Countdown

class libqtile.widget.**Countdown**(**args*, ***kwargs*)

A simple countdown timer text widget

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
date	datetime.datetime(2024, 4, 23, 13, 39, 51, 537622)	The datetime for the end of the countdown
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{D}d {H}h {M}m {S}s'	Format of the displayed text. Available variables:{D} == days, {H} == hours, {M} == minutes, {S} seconds.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1.0	Update interval in seconds for the clock

Available commands

Click to view the available commands for *Countdown*

8.16 CryptoTicker

class libqtile.widget.**CryptoTicker**(**args*, ***kwargs*)

A cryptocurrency ticker widget, data provided by the coinbase.com or the binance.com API. Defaults to displaying currency in whatever the current locale is. Examples:

display the average price of bitcoin in local currency widget.CryptoTicker()

display it in Euros: widget.CryptoTicker(currency="EUR")

or a different cryptocurrency! widget.CryptoTicker(crypto="ETH")

change the currency symbol: widget.CryptoTicker(currency="EUR", symbol="€")

display from Binance API widget.CryptoTicker(api="binance", currency="USDT")

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
api	'coinbase'	API that provides the data.
background	None	Widget background color
crypto	'BTC'	The cryptocurrency to display.
currency	''	The baseline currency that the value of the crypto is displayed in.
data	None	Post Data
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{crypto}: {symbol}{amount: 2f}'	Display string formatting.
headers	{}	Extra Headers
json	True	Is Json?
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
parse	None	Parse Function
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)

continues on next page

Table 3 – continued from previous page

key	default	description
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
symbol	''	The symbol for the baseline currency.
update_interval	600	Update interval in seconds, if none, the widget updates only once.
url	None	Url
user_agent	'Qtile'	Set the user agent
xml	False	Is XML?

Available commands

Click to view the available commands for *CryptoTicker*

8.17 CurrentLayout

class libqtile.widget.**CurrentLayout**(**args*, ***kwargs*)

Display the name of the current layout of the current group of the screen, the bar containing the widget, is on.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *CurrentLayout*

8.18 CurrentLayoutIcon

class libqtile.widget.**CurrentLayoutIcon**(**args*, ***kwargs*)

Display the icon representing the current layout of the current group of the screen on which the bar containing the widget is.

If you are using custom layouts, a default icon with question mark will be displayed for them. If you want to use custom icon for your own layout, for example, *FooGrid*, then create a file named "layout-foogrid.png" and place it in *~/.icons* directory. You can as well use other directories, but then you need to specify those directories in *custom_icon_paths* argument for this plugin.

The widget will look for icons with a *png* or *svg* extension.

The order of icon search is:

- dirs in *custom_icon_paths* config argument
- *~/.icons*

- built-in qtile icons

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
custom_icon_path	[]	List of folders where to search icons beforeusing built-in icons or icons in ~/.icons dir. This can also be used to providemissing icons for custom layouts. Defaults to empty list.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
scale	1	Scale factor relative to the bar height. Defaults to 1
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *CurrentLayoutIcon*

8.19 CurrentScreen

class libqtile.widget.**CurrentScreen**(**args*, ***kwargs*)

Indicates whether the screen this widget is on is currently active or not

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
active_color	'00ff00'	Color when screen is active
active_text	'A'	Text displayed when the screen is active
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
inactive_color	'ff0000'	Color when screen is inactive
inactive_text	'I'	Text displayed when the screen is inactive
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *CurrentScreen*

8.20 DF

class libqtile.widget.**DF**(**args*, ***kwargs*)

Disk Free Widget

By default the widget only displays if the space is less than warn_space.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{p} ({uf}{m}|{r:.0f}%)'	String format (p: partition, s: size, f: free space, uf: user free space, m: measure, r: ratio (uf/s))
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
measure	'G'	Measurement (G, M, B)
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
partition	'/'	the partition to check space
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	60	The update interval.
visible_on_warn	True	Only display if warning
warn_color	'ff0000'	Warning color
warn_space	2	Warning space in scale defined by the measure option.

Available commands

Click to view the available commands for *DF*

8.21 DoNotDisturb

class libqtile.widget.**DoNotDisturb**(**args*, ***kwargs*)

Displays Do Not Disturb status for notification server Dunst by default. Can be used with other servers by changing the poll command and mouse callbacks.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
disabled_icon	'O'	Icon that displays when do not disturb is disabled
enabled_icon	'X'	Icon that displays when do not disturb is enabled
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
poll_function	None	Function that returns the notification server status. Define the function on your configuration file and pass it like poll_function=my_func. Must return either true or false
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1	How often in seconds the text must update

Available commands

Click to view the available commands for *DoNotDisturb*

8.22 GenPollCommand

class libqtile.widget.**GenPollCommand**(**args*, ***kwargs*)

A generic text widget to display output from scripts or shell commands

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
cmd	None	command line as a string or list of arguments to execute
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
shell	False	run command through shell to enable piping and shell expansion
update_interval	60	update time in seconds

Available commands

Click to view the available commands for *GenPollCommand*

8.23 GenPollText

class libqtile.widget.**GenPollText**(**args*, ***kwargs*)

A generic text widget that polls using poll function to get the text

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format parameter of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
func	None	Poll Function
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	600	Update interval in seconds, if none, the widget updates only once.

Available commands

Click to view the available commands for *GenPollText*

8.24 GenPollUrl

class libqtile.widget.**GenPollUrl**(**args*, ***kwargs*)

A generic text widget that polls an url and parses it using parse function

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
data	None	Post Data
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
headers	{}	Extra Headers
json	True	Is Json?
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
parse	None	Parse Function
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	600	Update interval in seconds, if none, the widget updates only once.
url	None	Url
user_agent	'Qtile'	Set the user agent
xml	False	Is XML?

Available commands

Click to view the available commands for *GenPollUrl*

8.25 GmailChecker

class libqtile.widget.**GmailChecker**(**args*, ***kwargs*)

A simple gmail checker. If 'status_only_unseen' is True - set 'fmt' for one argument, ex. 'unseen: {0}'

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
display_fmt	'inbox[{0}], unseen[{1}]'	Display format
email_path	'INBOX'	email_path
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
password	None	password
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
status_only_unse	False	Only show unseen messages
update_interval	30	Update time in seconds.
username	None	username

Available commands

Click to view the available commands for *GmailChecker*

8.26 GroupBox

class libqtile.widget.**GroupBox**(**args*, ***kwargs*)

A widget that graphically displays the current group. All groups are displayed by their label. If the label of a group is the empty string that group will not be displayed.

Supported bar orientations: horizontal only

Configuration options

key	default	description
active	'FFFFFF'	Active group font colour
background	None	Widget background color
block_highlight_	None	Selected group font colour
borderwidth	3	Current group border width
center_aligned	True	center-aligned group box
disable_drag	False	Disable dragging and dropping of group names on widget
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you foo. To control what the widget outputs in the first place, use the `format` parameter of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
hide_unused	False	Hide groups that have no windows and that are not displayed on any screen.
highlight_color	['000000', '282828']	Active group highlight color when using 'line' highlight method.
highlight_method	'border'	Method of highlighting ('border', 'block', 'text', or 'line')Uses **_border* color settings
inactive	'404040'	Inactive group font colour
invert_mouse_whe	False	Whether to invert mouse wheel group movement
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
markup	False	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
other_current_sc	'404040'	Border or line colour for group on other screen when focused.
other_screen_bor	'404040'	Border or line colour for group on other screen when unfocused.
padding	None	Padding. Calculated if None.
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
rounded	True	To round or not to round box borders
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)

continues on next page

Table 4 – continued from previous page

key	default	description
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
spacing	None	Spacing between groups(if set to None, will be equal to margin_x)
this_current_scr	'215578'	Border or line colour for group on this screen when focused.
this_screen_bord	'215578'	Border or line colour for group on this screen when unfocused.
toggle	True	Enable toggling of group when clicking on same group name
urgent_alert_met	'border'	Method for alerting you of WM urgent hints (one of 'border', 'text', 'block', or 'line')
urgent_border	'FF0000'	Urgent border or line color
urgent_text	'FF0000'	Urgent group font color
use_mouse_wheel	True	Whether to use mouse wheel events
visible_groups	None	Groups that will be visible. If set to None or [], all groups will be visible.Visible groups are identified by name not by their displayed label.

Available commands

Click to view the available commands for *GroupBox*

8.27 HDDBusyGraph

class libqtile.widget.**HDDBusyGraph**(**args*, ***kwargs*)

Display HDD busy time graph

Parses /sys/block/<dev>/stat file and extracts overall device IO usage, based on io_ticks's value. See https://www.kernel.org/doc/Documentation/block/stat.txt

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
border_color	'215578'	Widget border color
border_width	2	Widget border width
device	'sda'	Block device to display info for
fill_color	'1667EB.3'	Fill color for linefill graph
frequency	1	Update frequency in seconds
graph_color	'18BAEB'	Graph color
line_width	3	Line width
margin_x	3	Margin X
margin_y	3	Margin Y
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
samples	100	Count of graph samples.
start_pos	'bottom'	Drawer starting position ('bottom'/'top')
type	'linefill'	'box', 'line', 'linefill'

Available commands

Click to view the available commands for *HDDBusyGraph*

8.28 HDDGraph

class libqtile.widget.**HDDGraph**(**args*, ***kwargs*)

Display HDD free or used space graph

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
border_color	'215578'	Widget border color
border_width	2	Widget border width
fill_color	'1667EB.3'	Fill color for linefill graph
frequency	1	Update frequency in seconds
graph_color	'18BAEB'	Graph color
line_width	3	Line width
margin_x	3	Margin X
margin_y	3	Margin Y
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
path	'/'	Partition mount point.
samples	100	Count of graph samples.
space_type	'used'	free/used
start_pos	'bottom'	Drawer starting position ('bottom'/'top')
type	'linefill'	'box', 'line', 'linefill'

Available commands

Click to view the available commands for *HDDGraph*

8.29 IdleRPG

class libqtile.widget.**IdleRPG**(**args*, ***kwargs*)

A widget for monitoring and displaying IdleRPG stats.

```
# display idlerpg stats for the player 'pants' on freenode's #idlerpg
widget.IdleRPG(url="http://xethron.lolhosting.net/xml.php?player=pants")
```

Widget requirements: xmltodict.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
data	None	Post Data
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'IdleRPG: {online} TTL: {ttl}'	Display format
headers	{}	Extra Headers
json	True	Is Json?
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
parse	None	Parse Function
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	600	Update interval in seconds, if none, the widget updates only once.
url	None	Url
user_agent	'Qtile'	Set the user agent
xml	False	Is XML?

Available commands

Click to view the available commands for *IdleRPG*

8.30 Image

class libqtile.widget.**Image**(**args*, ***kwargs*)

Display a PNG image on the bar

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
filename	None	Image filename. Can contain '~'
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
rotate	0.0	rotate the image in degrees counter-clockwise
scale	True	Enable/Disable image scaling

Available commands

Click to view the available commands for *Image*

8.31 ImapWidget

class libqtile.widget.**ImapWidget**(**args*, ***kwargs*)

Email IMAP widget

This widget will scan one of your imap email boxes and report the number of unseen messages present. I've configured it to only work with imap with ssl. Your password is obtained from the Gnome Keyring.

Writing your password to the keyring initially is as simple as (changing out <userid> and <password> for your userid and password):

1) create the file ~/.local/share/python_keyring/keyringrc.cfg with the following contents:

```
[backend]
default-keyring=keyring.backends.Gnome.Keyring
keyring-path=/home/<userid>/.local/share/keyring/
```

2) Execute the following python shell script once:

```
#!/usr/bin/env python3
import keyring
user = <userid>
password = <password>
keyring.set_password('imapwidget', user, password)
```

mbox names must include the path to the mbox (except for the default INBOX). So, for example if your mailroot is `~/Maildir`, and you want to look at the mailbox at HomeMail/fred, the mbox setting would be: `mbox="~/Maildir/HomeMail/fred"`. Note the nested sets of quotes! Labels can be whatever you choose, of course.

Widget requirements: keyring.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
label	'INBOX'	label for display
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mbox	'"INBOX"'	mailbox to fetch
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
server	None	email server name
update_interval	600	Update interval in seconds, if none, the widget updates only once.
user	None	email username

Available commands

Click to view the available commands for *ImapWidget*

8.32 KeyboardKbdd

class libqtile.widget.**KeyboardKbdd**(**args*, ***kwargs*)

Widget for changing keyboard layouts per window, using kbdd

kbdd should be installed and running, you can get it from: https://github.com/qnikst/kbdd

The widget also requires dbus-next.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
colours	None	foreground colour for each layouteither 'None' or a list of colours.example: ['ffffff', 'E6F0AF'].
configured_keybo	['us', 'ir']	your predefined list of keyboard layouts.example: ['us', 'ir', 'es']
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1	Update interval in seconds.

Available commands

Click to view the available commands for *KeyboardKbdd*

8.33 KeyboardLayout

class libqtile.widget.**KeyboardLayout**(**args*, ***kwargs*)

Widget for changing and displaying the current keyboard layout

To use this widget effectively you need to specify keyboard layouts you want to use (using "configured_keyboards") and bind function "next_keyboard" to specific keys in order to change layouts.

For example:

Key([mod], "space", lazy.widget["keyboardlayout"].next_keyboard(), desc="Next keyboard layout."),

When running Qtile with the X11 backend, this widget requires setxkbmap to be available. Xmodmap will also be used if .Xmodmap file is available.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
configured_keybo	['us']	A list of predefined keyboard layouts represented as strings. For example: ['us', 'us colemak', 'es', 'fr'].
display_map	{}	Custom display of layout. Key should be in format 'layout variant'. For example: {'us': 'us', 'lt sgs': 'sgs', 'ru phonetic': 'ru'}
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
option	None	string of setxkbmap option. Ex., 'compose:menu,grp_led:scroll'
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1	Update time in seconds.

Available commands

Click to view the available commands for *KeyboardLayout*

8.34 KhalCalendar

class libqtile.widget.**KhalCalendar**(**args*, ***kwargs*)

Khal calendar widget

This widget will display the next appointment on your Khal calendar in the qtile status bar. Appointments within the "reminder" time will be highlighted.

Widget requirements: dateutil.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'FFFF33'	default foreground color
lookahead	7	days to look ahead in the calendar
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
reminder_color	'FF0000'	color of calendar entries during reminder time
remindertime	10	reminder time in minutes
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	600	Update interval in seconds, if none, the widget updates only once.

Available commands

Click to view the available commands for *KhalCalendar*

8.35 LaunchBar

class libqtile.widget.**LaunchBar**(**args*, ***kwargs*)

This module defines a widget that displays icons to launch softwares or commands when clicked -- a launchbar. Only png icon files are displayed, not xpm because cairo doesn't support loading of xpm file. The order of displaying (from left to right) is in the order of the list.

If no icon was found for the name provided and if default_icon is set to None then the name is printed instead. If default_icon is defined then this icon is displayed instead.

To execute a software:

- ('thunderbird', 'thunderbird -safe-mode', 'launch thunderbird in safe mode')

To execute a python command in qtile, begin with by 'qshell:'

- ('/path/to/icon.png', 'qshell:self.qtile.shutdown()', 'logout from qtile')

Optional requirements: pyxdg for finding the icon path if it is not provided in the `progs` tuple.

Supported bar orientations: horizontal only

Configuration options

key	default	description
`background`	`None`	Widget background color
`default_icon`	`'/usr/share/icons/oxygen/256x256/mimetypes/application-x-ex png'`	Default icon not found
`font`	`'sans'`	Text font
`fontshadow`	`None`	Font shadow color, default is None (no shadow)
`fontsize`	`None`	Font pixel size. Calculated if None.
`foreground`	`'#ffffff'`	Text colour.
`icon_size`	`None`	Size of icons. `None` to fit to bar.
`mouse_callbacks`	`{}`	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
`padding`	`2`	Padding between icons
`padding_y`	`0`	Vertical adjustment for icons.
`progs`	`[]`	A list of tuples (software_name or icon_path, command_to_execute, comment), for example: [('thunderbird', 'thunderbird -safe-mode', 'launch thunderbird in safe mode'), ('/path/to/icon.png', 'qshell:self.qtile.shutdown()', 'logout from qtile')]
`text_only`	`False`	Don't use any icons.
`theme_path`	`None`	Path to icon theme to be used by pyxdg for icons. `None` will use default icon theme.

Available commands

Click to view the available commands for *LaunchBar*

8.36 Load

class libqtile.widget.**Load**(**args*, ***kwargs*)

A small widget to show the load averages of the system. Depends on psutil.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'Load({time}):{l 2f}'	The format in which to display the results.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1.0	The update interval for the widget

Available commands

Click to view the available commands for *Load*

8.37 Maildir

class libqtile.widget.**Maildir**(**args*, ***kwargs*)

A simple widget showing the number of new mails in maildir mailboxes

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
empty_color	None	Display color when no new mail is available
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
hide_when_empty	False	Whether not to display anything if the subfolder has no new mail
maildir_path	'~/Mail'	path to the Maildir folder
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
nonempty_color	None	Display color when new mail is available
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator	' '	the string to put between the subfolder strings.
sub_folders	[{'label': 'Home mail', 'path': 'INBOX'}, {'label': 'Home junk', 'path': 'spam'}]	List of subfolders to scan. Each subfolder is a dict of *path* and *label*.
subfolder_fmt	'{label}: {value}'	Display format for one subfolder
total	False	Whether or not to sum subfolders into a grand total. The first label will be used.
update_interval	600	Update interval in seconds, if none, the widget updates only once.

Available commands

Click to view the available commands for *Maildir*

8.38 Memory

class libqtile.widget.**Memory**(**args*, ***kwargs*)

Display memory/swap usage.

The following fields are available in the *format* string:

- `MemUsed`: Memory in use.
- `MemTotal`: Total amount of memory.
- `MemFree`: Amount of memory free.
- `Available`: Amount of memory available.
- `MemPercent`: Memory in use as a percentage.
- `Buffers`: Buffer amount.
- `Active`: Active memory.
- `Inactive`: Inactive memory.
- `Shmem`: Shared memory.
- `SwapTotal`: Total amount of swap.
- `SwapFree`: Amount of swap free.
- `SwapUsed`: Amount of swap in use.
- `SwapPercent`: Swap in use as a percentage.
- `mm`: Measure unit for memory.
- `ms`: Measure unit for swap.

Widget requirements: psutil.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{MemUsed: .0f}{mm}/ {MemTotal: .0f}{mm}'	Formatting for field names.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
measure_mem	'M'	Measurement for Memory (G, M, K, B)
measure_swap	'M'	Measurement for Swap (G, M, K, B)
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1.0	Update interval for the Memory

Available commands

Click to view the available commands for *Memory*

8.39 MemoryGraph

class libqtile.widget.**MemoryGraph**(**args*, ***kwargs*)

Displays a memory usage graph.

Widget requirements: psutil.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
border_color	'215578'	Widget border color
border_width	2	Widget border width
fill_color	'1667EB.3'	Fill color for linefill graph
frequency	1	Update frequency in seconds
graph_color	'18BAEB'	Graph color
line_width	3	Line width
margin_x	3	Margin X
margin_y	3	Margin Y
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
samples	100	Count of graph samples.
start_pos	'bottom'	Drawer starting position ('bottom'/'top')
type	'linefill'	'box', 'line', 'linefill'

Available commands

Click to view the available commands for *MemoryGraph*

8.40 Moc

class libqtile.widget.**Moc**(**args*, ***kwargs*)

A simple MOC widget.

Show the artist and album of now listening song and allow basic mouse control from the bar:

- toggle pause (or play if stopped) on left click;
- skip forward in playlist on scroll up;
- skip backward in playlist on scroll down.

MOC (http://moc.daper.net) should be installed.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
noplay_color	'cecece'	Text colour when not playing.
padding	None	Padding. Calculated if None.
play_color	'00ff00'	Text colour when playing.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	0.5	Update Time in seconds.

Available commands

Click to view the available commands for *Moc*

8.41 Mpd2

class libqtile.widget.**Mpd2**(**args*, ***kwargs*)

Mpd2 Object.

Parameters

status_format:
format string to display status

For a full list of values, see:
MPDClient.status() and MPDClient.currentsong()

https://musicpd.org/doc/protocol/command_reference.html#command_status https://musicpd.org/doc/protocol/tags.html

Default:

```
'{play_status} {artist}/{title} \
    [{repeat}{random}{single}{consume}{updating_db}]'

``play_status`` is a string from ``play_states`` dict

Note that the ``time`` property of the song renamed to ``fulltime``
to prevent conflicts with status information during formating.
```

idle_format:
format string to display status when no song is in queue.

Default:

```
'{play_status} {idle_message} \
    [{repeat}{random}{single}{consume}{updating_db}]'

Note that the ``artist`` key fallbacks to similar keys in specific␣
↪order.
(``artist`` -> ``albumartist`` -> ``performer`` ->
 -> ``composer`` -> ``conductor`` -> ``ensemble``)
```

idle_message:
text to display instead of song information when MPD is idle. (i.e. no song in queue)

Default:: "MPD IDLE"

undefined_value:
text to display when status key is undefined

Default:: "Undefined"

prepare_status:
dict of functions to replace values in status with custom characters.

`f(status, key, space_element) => str`

New functionality allows use of a dictionary of plain strings.

Default:

```
status_dict = {
    'repeat': 'r',
    'random': 'z',
    'single': '1',
    'consume': 'c',
    'updating_db': 'U'
}
```

format_fns:
A dict of functions to format the various elements.

'Tag': f(str) => str

Default:: { 'all': lambda s: cgi.escape(s) }

N.B. if 'all' is present, it is processed on every element of song_info
before any other formatting is done.

mouse_buttons:
A dict of mouse button numbers to actions

Widget requirements: python-mpd2_.
.. _python-mpd2: https://pypi.org/project/python-mpd2/

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
color_progress	None	Text color to indicate track progress.
command	<function default_cmd at 0x7fa5f8125ab0>	command to be executed by mapped mouse button.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format parameter of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format_fns	{'all': <function escape at 0x7fa5fab513f0>}	Dictionary of format methods
host	'localhost'	Host of mpd server
idle_format	'{play_status} {idle_message}[{	format for status when mpd has no playlist.
idle_message	'MPD IDLE'	text to display when mpd is idle.
idletimeout	5	MPDClient idle command timeout
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_buttons	{1: 'toggle', 3: 'stop', 4: 'previous', 5: 'next'}	b_num -> action.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
no_connection	'No connection'	Text when mpd is disconnected
padding	None	Padding. Calculated if None.
password	None	Password for auth on mpd server
play_states	{'pause': '', 'play': '', 'stop': ''}	Play state mapping
port	6600	Port of mpd server

continues on next page

Table 5 – continued from previous page

key	default	description
prepare_status	{'consume': 'c', 'random': 'z', 'repeat': 'r', 'single': '1', 'updating_db': 'U'}	characters to show the status of MPD
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
space	'-'	Space keeper
status_format	'{play_status} {artist}/ {title} [{repeat}{random	format for displayed song info.
timeout	30	MPDClient timeout
undefined_value	'Undefined'	text to display when status key is undefined.
update_interval	1	Interval of update widget

Available commands

Click to view the available commands for *Mpd2*

8.42 Mpris2

class libqtile.widget.**Mpris2**(**args*, ***kwargs*)

An MPRIS 2 widget

A widget which displays the current track/artist of your favorite MPRIS player. This widget scrolls the text if neccessary and information that is displayed is configurable.

The widget relies on players broadcasting signals when the metadata or playback status changes. If you are getting inconsistent results then you can enable background polling of the player by setting the *poll_interval* parameter. This is disabled by default.

Basic mouse controls are also available: button 1 = play/pause, scroll up = next track, scroll down = previous track.

Widget requirements: dbus-next.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
display_metadata	['xesam:title', 'xesam:album', 'xesam:artist']	(Deprecated) Which metadata identifiers to display.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{xesam:title} – {xesam:album} – {xesam:artist}'	Format string for displaying metadata. See http://www.freedesktop.org/wiki/Specifications/mpris-spec/metadata/#index5h3 for available values
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
name	'audacious'	Name of the MPRIS widget.
no_metadata_text	'No metadata for current track'	Text to show when track has no metadata
objname	None	DBUS MPRIS 2 compatible player identifier- Find it out with dbus-monitor - Also see: http://specifications.freedesktop.org/mpris-spec/latest/#Bus-Name-Policy. None will listen for notifications from all MPRIS2 compatible players.
padding	None	Padding. Calculated if None.
paused_text	'Paused: {track}'	Text to show when paused
playing_text	'{track}'	Text to show when playing
poll_interval	0	Periodic background polling interval of player (0 to disable polling).
scroll	True	Whether text should scroll.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator	', '	Separator for metadata fields that are a list.
stop_pause_text	None	(Deprecated) Optional text to display when in the stopped/paused state
stopped_text	''	Text to show when stopped

Available commands

Click to view the available commands for *Mpris2*

8.43 Net

class libqtile.widget.**Net**(**args*, ***kwargs*)

Displays interface down and up speed

Widget requirements: psutil.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{interface}: {down:6. 2f}{down_suffix: 2f}{up_suffix:<2	Display format of down/upload/total speed of given interfaces
interface	None	List of interfaces or single NIC as string to monitor, None to display all active NICs combined
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
prefix	None	Use a specific prefix for the unit of the speed.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1	The update interval.
use_bits	False	Use bits instead of bytes per second?

Available commands

Click to view the available commands for *Net*

8.44 NetGraph

class libqtile.widget.**NetGraph**(**args*, ***kwargs*)

Display a network usage graph.

Widget requirements: psutil.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
bandwidth_type	'down'	down(load)/up(load)
border_color	'215578'	Widget border color
border_width	2	Widget border width
fill_color	'1667EB.3'	Fill color for linefill graph
frequency	1	Update frequency in seconds
graph_color	'18BAEB'	Graph color
interface	'auto'	Interface to display info for ('auto' for detection)
line_width	3	Line width
margin_x	3	Margin X
margin_y	3	Margin Y
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
samples	100	Count of graph samples.
start_pos	'bottom'	Drawer starting position ('bottom'/'top')
type	'linefill'	'box', 'line', 'linefill'

Available commands

Click to view the available commands for *NetGraph*

8.45 Notify

class libqtile.widget.**Notify**(**args*, ***kwargs*)

A notify widget

This widget can handle actions provided by notification clients. However, only the default action is supported, so if a client provides multiple actions then only the default (first) action can be invoked. Some programs will provide their own notification windows if the notification server does not support actions, so if you want your notifications to handle more than one action then specify False for the action option to disable all action handling. Unfortunately we cannot specify the capability for exactly one action.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
action	True	Enable handling of default action upon right click
audiofile	None	Audiofile played during notifications
background	None	Widget background color
background_low	'444444'	Background low priority colour
background_urgen	'440000'	Background urgent priority colour
default_timeout	10	Default timeout (seconds) for normal notifications
default_timeout_	5	Default timeout (seconds) for low urgency notifications.
default_timeout_	None	Default timeout (seconds) for urgent notifications
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
foreground_low	'dddddd'	Foreground low priority colour
foreground_urgen	'ff0000'	Foreground urgent priority colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
parse_text	None	Function to parse and modify notifications. e.g. function in config that removes line returns:def my_func(text) return text.replace('n', '')then set option parse_text=my_func
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *Notify*

8.46 NvidiaSensors

class libqtile.widget.**NvidiaSensors**(**args*, ***kwargs*)

Displays temperature, fan speed and performance level Nvidia GPU.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format parameter of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
foreground_alert	'ff0000'	Foreground colour alert
format	'{temp}°C'	Display string format. Three options available: {temp} - temperature, {fan_speed} and {perf} - performance level
gpu_bus_id	''	GPU's Bus ID, ex: 01:00.0. If leave empty will display all available GPU's
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
threshold	70	If the current temperature value is above, then change to foreground_alert colour
update_interval	2	Update interval in seconds.

Available commands

Click to view the available commands for *NvidiaSensors*

8.47 OpenWeather

class libqtile.widget.**OpenWeather**(**args*, ***kwargs*)

A weather widget, data provided by the OpenWeather API.

Some format options:

- location_city
- location_cityid
- location_country
- location_lat
- location_long
- weather
- weather_details
- units_temperature
- units_wind_speed
- isotime
- humidity
- pressure
- sunrise
- sunset
- temp
- visibility
- wind_speed
- wind_deg
- wind_direction
- main_feels_like
- main_temp_min
- main_temp_max
- clouds_all
- icon

Icon support is available but you will need a suitable font installed. A default icon mapping is provided (`OpenWeather.symbols`) but changes can be made by setting `weather_symbols`. Available icon codes can be viewed here: https://openweathermap.org/weather-conditions#Icon-list

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
app_key	'7834197c2338888	Open Weather access key. A default is provided, butn for prolonged use obtaining your own is suggested:n https://home.openweathermap.org/users/sign_up
background	None	Widget background color
cityid	None	City ID. Can be looked up on e.g.:n https://openweathermap.org/findn Takes precedence over location and coordinates.n Note that this is not equal to a WOEID.
coordinates	None	Dictionary containing latitude and longituden Example: coordinates={"longitude": "77.22",n "latitude": "28.67"}
data	None	Post Data
dateformat	'%Y-%m-%d '	Format for dates, defaults to ISO.n For details see: https://docs.python.org/3/library/time.html#time.strftime
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{location_city} {main_temp} °{units_temperat {humidity}% {weather_details	Display format
headers	{}	Extra Headers
json	True	Is Json?
language	'en'	Language of response. List of languages supported cann be seen at: https://openweathermap.org/current undern Multilingual support
location	None	Name of the city. Country name can be appendedn like cambridge,NZ. Takes precedence over zip-code.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
metric	True	True to use metric/C, False to use imperial/F
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
parse	None	Parse Function
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.

continues on next page

Table 6 – continued from previous page

key	default	description
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
timeformat	'%H:%M'	Format for times, defaults to ISO.n For details see: https://docs.python.org/3/library/time.html#time.strftime
update_interval	600	Update interval in seconds, if none, the widget updates only once.
url	None	Url
user_agent	'Qtile'	Set the user agent
weather_symbols	{}	Dictionary of weather symbols. Can be used to override default symbols.
xml	False	Is XML?
zip	None	Zip code (USA) or "zip code,country code" forn other countries. E.g. 12345,NZ. Takes precedence overn coordinates.

Available commands

Click to view the available commands for *OpenWeather*

8.48 Pomodoro

class libqtile.widget.**Pomodoro**(**args*, ***kwargs*)

Pomodoro technique widget

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
color_active	'00ff00'	Colour then pomodoro is running
color_break	'ffff00'	Colour then it is break time
color_inactive	'ff0000'	Colour then pomodoro is inactive
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
length_long_brea	15	Length of a long break in minutes
length_pomodori	25	Length of one pomodori in minutes
length_short_bre	5	Length of a short break in minutes
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.

continues on next page

Table 7 – continued from previous page

key	default	description
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
notification_on	True	Turn notifications on
num_pomodori	4	Number of pomodori to do in a cycle
padding	None	Padding. Calculated if None.
prefix_active	''	Prefix then app is active
prefix_break	'B '	Prefix during short break
prefix_inactive	'POMODORO'	Prefix when app is inactive
prefix_long_brea	'LB '	Prefix during long break
prefix_paused	'PAUSE'	Prefix during pause
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1	Update interval in seconds, if none, the widget updates whenever the event loop is idle.

Available commands

Click to view the available commands for *Pomodoro*

8.49 Prompt

class libqtile.widget.**Prompt**(**args*, ***kwargs*)

A widget that prompts for user input

Input should be started using the .start_input() method on this class.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
bell_style	'audible'	Alert at the begin/end of the command history. Possible values: 'audible' (X11 only), 'visual' and None.
cursor	True	Show a cursor
cursor_color	'bef098'	Color for the cursor and text over it.
cursorblink	0.5	Cursor blink rate. 0 to disable.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
ignore_dups_hist	False	Don't store duplicates in history
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
max_history	100	Commands to keep in history. 0 for no limit.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
prompt	'{prompt}: '	Text displayed at the prompt
record_history	True	Keep a record of executed commands
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
visual_bell_colo	'ff0000'	Color for the visual bell (changes prompt background).
visual_bell_time	0.2	Visual bell duration (in seconds).

Available commands

Click to view the available commands for *Prompt*

8.50 PulseVolume

class libqtile.widget.**PulseVolume**(**args*, ***kwargs*)

Volume widget for systems using PulseAudio.

The widget connects to the PulseAudio server by using the libpulse library and so should be updated virtually instantly rather than needing to poll the volume status regularly (NB this means that the `update_interval` parameter serves no purpose for this widget).

The widget relies on the pulsectl_asyncio library to access the libpulse bindings. If you are using python 3.11 you must use `pulsectl_asyncio >= 1.0.0`.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
cardid	None	Card Id
channel	'Master'	Channel
check_mute_comma	None	Command to check mute status
check_mute_strin	'[off]'	String expected from check_mute_command when volume is muted.When the output of the command matches this string, theaudio source is treated as muted.
device	'default'	Device Name
emoji	False	Use emoji to display volume states, only if `theme_path` is not set.The specified font needs to contain the correct unicode characters.
emoji_list	['', '', '', '']	List of emojis/font-symbols to display volume states, only if `emoji` is set. List contains 4 symbols, from lowest volume to highest.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
get_volume_comma	None	Command to get the current volume. The expected output should include 1-3 numbers and a `%` sign.
limit_max_volume	False	Limit maximum volume to 100%
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
mute_command	None	Mute command
padding	3	Padding left and right. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)

continues on next page

Table 8 – continued from previous page

key	default	description
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
step	2	Volume change for up an down commands in percentage.Only used if volume_up_command and volume_down_command are not set.
theme_path	None	Path of the icons
update_interval	0.2	Update time in seconds.
volume_app	None	App to control volume
volume_down_comm	None	Volume down command
volume_up_comman	None	Volume up command

Available commands

Click to view the available commands for *PulseVolume*

8.51 QuickExit

class libqtile.widget.**QuickExit**(**args*, ***kwargs*)

A button to shut down Qtile. When clicked, a countdown starts. Clicking the button again stops the countdown and prevents Qtile from shutting down.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
countdown_format	'[{} seconds]'	The text displayed when counting down.
countdown_start	5	The number to count down from.
default_text	'[shutdown]'	The text displayed on the button.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you foo. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
timer_interval	1	The countdown interval.

Available commands

Click to view the available commands for *QuickExit*

8.52 ScreenSplit

class libqtile.widget.**ScreenSplit**(**args*, ***kwargs*)

A simple widget to show the name of the current split and layout for the `ScreenSplit` layout.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Text font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font pixel size. Calculated if None.
foreground	'#ffffff'	Foreground colour.
format	'{split_name} ({layout})'	Format string.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding left and right. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *ScreenSplit*

8.53 Sep

class libqtile.widget.**Sep**(**args*, ***kwargs*)

A visible widget separator

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
foreground	'888888'	Separator line colour.
linewidth	1	Width of separator line.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	2	Padding on either side of separator.
size_percent	80	Size as a percentage of bar size (0-100).

Available commands

Click to view the available commands for *Sep*

8.54 She

class libqtile.widget.**She**(**args*, ***kwargs*)

Widget to display the Super Hybrid Engine status

Can display either the mode or CPU speed on eeepc computers.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
device	'/sys/devices/platform/eeepc/cpufv'	sys path to cpufv
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'speed'	Type of info to display "speed" or "name"
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	0.5	Update Time in seconds.

Available commands

Click to view the available commands for *She*

8.55 Spacer

class libqtile.widget.**Spacer**(**args*, ***kwargs*)

Just an empty space on the bar

Often used with length equal to bar.STRETCH to push bar widgets to the right or bottom edge of the screen.

Parameters

length
Length of the widget. Can be either bar.STRETCH or a length in pixels.

width
DEPRECATED, same as `length`.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
`background`	`None`	Widget background color
`mouse_callbacks`	`{}`	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.

Available commands

Click to view the available commands for *Spacer*

8.56 StatusNotifier

class libqtile.widget.**StatusNotifier**(**args*, ***kwargs*)

A 'system tray' widget using the freedesktop StatusNotifierItem specification.

As per the specification, app icons are first retrieved from the user's current theme. If this is not available then the app may provide its own icon. In order to use this functionality, users are recommended to install the pyxdg module to support retrieving icons from the selected theme. If the icon specified by StatusNotifierItem can not be found in the user's current theme and no other icons are provided by the app, a fallback icon is used.

Left-clicking an icon will trigger an activate event.

Note: Context menus are not currently supported by the official widget. However, a modded version of the widget which provides basic menu support is available from elParaguayo's qtile-extras repo.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
`background`	`None`	Widget background color
`icon_size`	`16`	Icon width
`icon_theme`	`None`	Name of theme to use for app icons
`mouse_callbacks`	`{}`	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
`padding`	`3`	Padding between icons

Available commands

Click to view the available commands for *StatusNotifier*

8.57 StockTicker

class libqtile.widget.**StockTicker**(**args*, ***kwargs*)

A stock ticker widget, based on the alphavantage API. Users must acquire an API key from https://www.alphavantage.co/support/#api-key

The widget defaults to the TIME_SERIES_INTRADAY API function (i.e. stock symbols), but arbitrary Alpha Vantage API queries can be made by passing extra arguments to the constructor.

```
# Display AMZN
widget.StockTicker(apikey=..., symbol="AMZN")

# Display BTC
widget.StockTicker(
    apikey=..., function="DIGITAL_CURRENCY_INTRADAY", symbol="BTC", market="USD"
)
```

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
data	None	Post Data
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
func	'TIME_SERIES_INT	The default API function to query
function	'TIME_SERIES_INT	DEPRECATED: Use *func*.
headers	{}	Extra Headers
interval	'1min'	The default latency to query
json	True	Is Json?
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
parse	None	Parse Function
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	600	Update interval in seconds, if none, the widget updates only once.
url	None	Url
user_agent	'Qtile'	Set the user agent
xml	False	Is XML?

Available commands

Click to view the available commands for *StockTicker*

8.58 SwapGraph

class libqtile.widget.**SwapGraph**(**args*, ***kwargs*)

Display a swap info graph.

Widget requirements: psutil.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
border_color	'215578'	Widget border color
border_width	2	Widget border width
fill_color	'1667EB.3'	Fill color for linefill graph
frequency	1	Update frequency in seconds
graph_color	'18BAEB'	Graph color
line_width	3	Line width
margin_x	3	Margin X
margin_y	3	Margin Y
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
samples	100	Count of graph samples.
start_pos	'bottom'	Drawer starting position ('bottom'/'top')
type	'linefill'	'box', 'line', 'linefill'

Available commands

Click to view the available commands for *SwapGraph*

8.59 Systray

class libqtile.widget.**Systray**(**args*, ***kwargs*)

A widget that manages system tray.

Only one Systray widget is allowed. Adding additional Systray widgets will result in a ConfigError.

Note: Icons will not render correctly where the bar/widget is drawn with a semi-transparent background. Instead, icons will be drawn with a transparent background.

If using this widget it is therefore recommended to use a fully opaque background colour or a fully transparent one.

Supported bar orientations: horizontal and vertical

Only available on the following backends: x11

Configuration options

key	default	description
background	None	Widget background color
icon_size	20	Icon width
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	5	Padding between icons

Available commands

Click to view the available commands for *Systray*

8.60 TaskList

class libqtile.widget.**TaskList**(**args*, ***kwargs*)

Displays the icon and name of each window in the current group

Contrary to WindowTabs this is an interactive widget. The window that currently has focus is highlighted.

Optional requirements: pyxdg is needed to use theme icons and to display icons on Wayland.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
border	'215578'	Border colour
borderwidth	2	Current group border width
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
highlight_method	'border'	Method of highlighting (one of 'border' or 'block') Uses *_border* color settings
icon_size	None	Icon size. (Calculated if set to None. Icons are hidden if set to 0.)
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
markup_floating	None	Text markup of the floating window state. Supports pangomarkup with markup=True.e.g., "{}" or "{}"
markup_focused	None	Text markup of the focused window state. Supports pangomarkup with markup=True.e.g., "{}" or "{}"
markup_focused_f	None	Text markup of the focused and floating window state. Supports pangomarkup with markup=True.e.g., "{}" or "{}"
markup_maximized	None	Text markup of the maximized window state. Supports pangomarkup with markup=True.e.g., "{}" or "{}"

continues on next page

Table 9 – continued from previous page

key	default	description
markup_minimized	None	Text markup of the minimized window state. Supports pangomarkup with markup=True.e.g., "{}" or "{}"
markup_normal	None	Text markup of the normal window state. Supports pangomarkup with markup=True.e.g., "{}" or "{}"
max_title_width	None	Max size in pixels of task title.(if set to None, as much as available.)
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	3	Padding inside the box
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
parse_text	None	Function to parse and modify window names. e.g. function in config that removes excess strings from window name: def my_func(text) for string in [" - Chromium", " - Firefox"]: text = text.replace(string, "") return textthen set option parse_text=my_func
rounded	True	To round or not to round borders
spacing	None	Spacing between tasks.(if set to None, will be equal to margin_x)
stretch	True	Widget fills available space in bar. Set to *False* to limit widget width to size of its contents.
theme_mode	None	When to use theme icons. *None* = never, *preferred* = use if available, *fallback* = use if app does not provide icon directly. *preferred* and *fallback* have identical behaviour on Wayland.
theme_path	None	Path to icon theme to be used by pyxdg for icons. None will use default icon theme.
title_width_meth	None	Method to compute the width of task title. (None, 'uniform'.)Defaults to None, the normal behaviour.
txt_floating	'V '	Text representation of the floating window state. e.g., "V " or " "
txt_maximized	'[] '	Text representation of the maximized window state. e.g., "[] " or " "
txt_minimized	'_ '	Text representation of the minimized window state. e.g., "_ " or " "
unfocused_border	None	Border color for unfocused windows. Affects only hightlight_method 'border' and 'block'. Defaults to None, which means no special color.
urgent_alert_met	'border'	Method for alerting you of WM urgent hints (one of 'border' or 'text')
urgent_border	'FF0000'	Urgent border color
window_name_loca	False	Whether to show the location of the window in the title.
window_name_loca	0	The offset given to the window location

Available commands

Click to view the available commands for *TaskList*

8.61 TextBox

class libqtile.widget.**TextBox**(**args*, ***kwargs*)

A flexible textbox that can be updated from bound keys, scripts, and qshell.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format parameter of the widget (if it has one).
font	'sans'	Text font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font pixel size. Calculated if None.
foreground	'#ffffff'	Foreground colour.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding left and right. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *TextBox*

8.62 ThermalSensor

class libqtile.widget.**ThermalSensor**(**args*, ***kwargs*)

Widget to display temperature sensor information

For using the thermal sensor widget you need to have lm-sensors installed. You can get a list of the tag_sensors executing "sensors" in your terminal. Then you can choose which you want, otherwise it will display the first available.

Widget requirements: psutil.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
foreground_alert	'ff0000'	Foreground colour alert
format	'{temp:.1f}{unit}'	Display string format. Three options available: {temp} - temperature, {tag} - tag of the temperature sensor, and {unit} - °C or °F
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
metric	True	True to use metric/C, False to use imperial/F
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
tag_sensor	None	Tag of the temperature sensor. For example: "temp1" or "Core 0"
threshold	70	If the current temperature value is above, then change to foreground_alert colour
update_interval	2	Update interval in seconds

Available commands

Click to view the available commands for *ThermalSensor*

8.63 ThermalZone

class libqtile.widget.**ThermalZone**(**args*, ***kwargs*)

Thermal zone widget.

This widget was made to read thermal zone files and transform values to human readable format. You can set zone parameter to any standard thermal zone file from /sys/class/thermal directory.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
crit	70	Critical temperature level
fgcolor_crit	'ff0000'	Font color on critical values
fgcolor_high	'ffaa00'	Font color on high values
fgcolor_normal	'ffffff'	Font color on normal values
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you foo. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{temp}°C'	Display format
format_crit	'{temp}°C CRIT!'	Critical display format
hidden	False	Set True to only show if critical value reached
high	50	High themperature level
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	2.0	Update interval
zone	'/sys/class/thermal/thermal_zone0/temp'	Thermal zone

Available commands

Click to view the available commands for *ThermalZone*

8.64 Volume

class libqtile.widget.**Volume**(**args*, ***kwargs*)

Widget that display and change volume

By default, this widget uses `amixer` to get and set the volume so users will need to make sure this is installed. Alternatively, users may set the relevant parameters for the widget to use a different application.

If theme_path is set it draw widget as icons.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
cardid	None	Card Id
channel	'Master'	Channel
check_mute_comma	None	Command to check mute status
check_mute_strin	'[off]'	String expected from check_mute_command when volume is muted.When the output of the command matches this string, theaudio source is treated as muted.
device	'default'	Device Name
emoji	False	Use emoji to display volume states, only if `theme_path` is not set.The specified font needs to contain the correct unicode characters.
emoji_list	['', '', '', '']	List of emojis/font-symbols to display volume states, only if `emoji` is set. List contains 4 symbols, from lowest volume to highest.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you foo. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
get_volume_comma	None	Command to get the current volume. The expected output should include 1-3 numbers and a `%` sign.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
mute_command	None	Mute command
padding	3	Padding left and right. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end

continues on next page

Table 10 – continued from previous page

key	default	description
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
step	2	Volume change for up an down commands in percentage.Only used if volume_up_command and volume_down_command are not set.
theme_path	None	Path of the icons
update_interval	0.2	Update time in seconds.
volume_app	None	App to control volume
volume_down_comm	None	Volume down command
volume_up_comman	None	Volume up command

Available commands

Click to view the available commands for *Volume*

8.65 Wallpaper

class libqtile.widget.**Wallpaper**(**args*, ***kwargs*)

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
directory	'~/Pictures/wallpapers/'	Wallpaper Directory
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
label	None	Use a fixed label instead of image name.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
option	'fill'	How to fit the wallpaper when wallpaper_command isNone. None, 'fill' or 'stretch'.
padding	None	Padding. Calculated if None.
random_selection	False	If set, use random initial wallpaper and randomly cycle through the wallpapers.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
wallpaper	None	Wallpaper
wallpaper_comman	['feh', '--bg-fill']	Wallpaper command. If None, thewallpaper will be painted without the use of a helper.

Available commands

Click to view the available commands for *Wallpaper*

8.66 WidgetBox

class libqtile.widget.**WidgetBox**(**args*, ***kwargs*)

A widget to declutter your bar.

WidgetBox is a widget that hides widgets by default but shows them when the box is opened.

Widgets that are hidden will still update etc. as if they were on the main bar.

Button clicks are passed to widgets when they are visible so callbacks will work.

Widgets in the box also remain accessible via command interfaces.

Widgets can only be added to the box via the configuration file. The widget is configured by adding widgets to the "widgets" parameter as follows:

```
widget.WidgetBox(widgets=[
    widget.TextBox(text="This widget is in the box"),
    widget.Memory()
    ]
),
```

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
close_button_loc	'left'	Location of close button when box open ('left' or 'right')
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
start_opened	False	Spawn the box opened
text_closed	'[<]'	Text when box is closed
text_open	'[>]'	Text when box is open
widgets	[]	A list of widgets to include in the box

Available commands

Click to view the available commands for *WidgetBox*

8.67 WindowCount

class libqtile.widget.**WindowCount**(**args*, ***kwargs*)

A simple widget to display the number of windows in the current group of the screen on which the widget is.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Text font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font pixel size. Calculated if None.
foreground	'#ffffff'	Foreground colour.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding left and right. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
show_zero	False	Show window count when no windows
text_format	'{num}'	Format for message

Available commands

Click to view the available commands for *WindowCount*

8.68 WindowName

class libqtile.widget.**WindowName**(**args*, ***kwargs*)

Displays the name of the window that currently has focus

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
empty_group_stri	' '	string to display when no windows are focused on current group
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
for_current_scre	False	instead of this bars screen use currently active screen
foreground	'ffffff'	Foreground colour
format	'{state}{name}'	format of the text
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
parse_text	None	Function to parse and modify window names. e.g. function in config that removes excess strings from window name: def my_func(text) for string in [" - Chromium", " - Firefox"]: text = text.replace(string, "") return textthen set option parse_text=my_func
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step

Available commands

Click to view the available commands for *WindowName*

8.69 WindowTabs

class libqtile.widget.**WindowTabs**(**args*, ***kwargs*)

Displays the name of each window in the current group. Contrary to TaskList this is not an interactive widget. The window that currently has focus is highlighted.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
parse_text	<function WindowTabs. <lambda> at 0x7fa5f8050790>	Function to modify window names. It must accept a string argument (original window name) and return a string with the modified name.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
selected	('', '')	Selected task indicator
separator	' | '	Task separator text.

Available commands

Click to view the available commands for *WindowTabs*

8.70 Wlan

class libqtile.widget.**Wlan**(**args*, ***kwargs*)

Displays Wifi SSID and quality.

Widget requirements: iwlib.

Supported bar orientations: horizontal only

Configuration options

key	default	description
background	None	Widget background color
disconnected_mes	'Disconnected'	String to show when the wlan is diconnected.
ethernet_interfa	'eth0'	The ethernet interface to monitor, NOTE: If you do not have a wlan device in your system, ethernet functionality will not work, use the Net widget instead
ethernet_message	'eth'	String to show when ethernet is being used
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{essid} {quality}/70'	Display format. For percents you can use "{essid} {percent:2.0%}"
interface	'wlan0'	The interface to monitor
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
update_interval	1	The update interval.
use_ethernet	False	Activate or deactivate checking for ethernet when no wlan connection is detected

Available commands

Click to view the available commands for *Wlan*

8.71 Wttr

class libqtile.widget.**Wttr**(**args*, ***kwargs*)

Display weather widget provided by wttr.in.

To specify your own custom output format, use the special %-notation (example: 'My_city: %t(%f), wind: %w'):

- %c Weather condition,
- %C Weather condition textual name,
- %h Humidity,
- %t Temperature (Actual),
- %f Temperature (Feels Like),
- %w Wind,
- %l Location,
- %m Moonphase ,
- %M Moonday,
- %p precipitation (mm),
- %P pressure (hPa),
- %D Dawn !,
- %S Sunrise !,
- %z Zenith !,
- %s Sunset !,
- %d Dusk !. (!times are shown in the local timezone)

Add the character ~ at the beginning to get weather for some special location: `~Vostok Station` or `~Eiffel Tower`.

Also can use IP-addresses (direct) or domain names (prefixed with `@`) to specify a location: `@github.com`, `123.456.678.123`

Specify multiple locations as dictionary

```
location={
    'Minsk': 'Minsk',
    '64.127146,-21.873472': 'Reykjavik',
}
```

Cities will change randomly every update.

Supported bar orientations: horizontal and vertical

Configuration options

key	default	description
background	None	Widget background color
data	None	Post Data
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'3'	Display text format. Choose presets in range 1-4 (Ex. "1") or build your own custom output format, use the special %-notation. See https://github.com/chubin/wttr.in#one-line-output
headers	{}	Extra Headers
json	False	Is Json?
lang	'en'	Display text language. List of supported languages https://wttr.in/:translation
location	{}	Dictionary. Key is a city or place name, or GPS coordinates. Value is a display name. If the dictionary is empty, the location will be determined based on your IP address.
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	None	Padding. Calculated if None.
parse	None	Parse Function
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
units	'm'	'm' - metric, 'M' - show wind speed in m/s, 'u' - United States units
update_interval	600	Update interval in seconds. Recommendation: if you want to display multiple locations alternately, maybe set a smaller interval, ex. 30.
url	None	Url
user_agent	'Qtile'	Set the user agent
xml	False	Is XML?

Available commands

Click to view the available commands for *Wttr*

CHAPTER

NINE

BUILT-IN HOOKS

subscribe.**addgroup**()

Called when a new group is added

Arguments

- name of new group

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.addgroup
def group_added(group_name):
    send_notification("qtile", f"New group added: {group_name}")
```

subscribe.**changegroup**()

Called whenever a group change occurs.

The following changes will result in this hook being fired: 1) New group added (unlike `addgroup`, no group name is passed with this hook) 2) Group deleted (unlike `delgroup`, no group name is passed with this hook) 3) Groups order is changed 4) Group is renamed

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.changegroup
def change_group():
    send_notification("qtile", "Change group event")
```

subscribe.**client_focus**()

Called whenever focus moves to a client window

Arguments

- `Window` object of the new focus.

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.client_focus
def client_focus(client):
    send_notification("qtile", f"{client.name} has been focused")
```

subscribe.**client_killed**()

Called after a client has been unmanaged

Arguments

- `Window` object of the killed window.

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.client_killed
def client_killed(client):
    send_notification("qtile", f"{client.name} has been killed")
```

subscribe.**client_managed**()

Called after Qtile starts managing a new client

Called after a window is assigned to a group, or when a window is made static. This hook is not called for internal windows.

Arguments

- `Window` object of the managed window

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.client_managed
def client_managed(client):
    send_notification("qtile", f"{client.name} has been managed by qtile")
```

subscribe.**client_mouse_enter**()

Called when the mouse enters a client

Arguments

- `Window` of window entered

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.client_mouse_enter
def client_mouse_enter(client):
    send_notification("qtile", f"Mouse has entered {client.name}")
```

subscribe.**client_name_updated**()

Called when the client name changes

Arguments

- Window of client with updated name

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.client_name_updated
def client_name_updated(client):
    send_notification(
        "qtile",
        f"Client's has been updated to {client.name}"
    )
```

subscribe.**client_new**()

Called before Qtile starts managing a new client

Use this hook to declare windows static, or add them to a group on startup. This hook is not called for internal windows.

Arguments

- Window object

Example:

```
from libqtile import hook

@hook.subscribe.client_new
def new_client(client):
    if client.name == "xterm":
        client.togroup("a")
    elif client.name == "dzen":
        client.static(0)
```

subscribe.**client_urgent_hint_changed**()

Called when the client urgent hint changes

Arguments

- Window of client with hint change

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.client_urgent_hint_changed
def client_urgency_change(client):
    send_notification(
        "qtile",
        f"{client.name} has changed its urgency state"
    )
```

subscribe.**current_screen_change**()

Called when the current screen (i.e. the screen with focus) changes

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.current_screen_change
def screen_change():
    send_notification("qtile", "Current screen change detected.")
```

subscribe.**delgroup**()

Called when group is deleted

Arguments

- name of deleted group

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.delgroup
def group_deleted(group_name):
    send_notification("qtile", f"Group deleted: {group_name}")
```

subscribe.**enter_chord**()

Called when key chord begins

Note: if you only want to use this chord to display the chord name then you should use the `Chord` widget.

Arguments

- name of chord(mode)

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.enter_chord
def enter_chord(chord_name):
    send_notification("qtile", "Started {chord_name} key chord.")
```

subscribe.**float_change**()

Called when a change in float state is made (e.g. toggle floating, minimised and fullscreen states)

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.float_change
def float_change():
    send_notification("qtile", "Window float state changed.")
```

subscribe.**focus_change**()

Called when focus is changed, including moving focus between groups or when focus is lost completely (i.e. when a window is closed.)

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.focus_change
def focus_changed():
    send_notification("qtile", "Focus changed.")
```

subscribe.**group_window_add**()

Called when a new window is added to a group

Arguments

- `Group` receiving the new window
- `Window` added to the group

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.group_window_add
def group_window_add(group, window):
    send_notification("qtile", f"Window {window.name} added to {group.name}")
```

subscribe.**layout_change**()

Called on layout change event (including when a new group is displayed on the screen)

Arguments

- layout object for new layout
- group object on which layout is changed

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.layout_change
def layout_change(layout, group):
```

(continues on next page)

(continued from previous page)

```
    send_notification(
        "qtile",
        f"{layout.name} is now on group {group.name}"
    )
```

subscribe.**leave_chord**()

Called when key chord ends

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.leave_chord
ded leave_chord():
    send_notification("qtile", "Key chord exited")
```

subscribe.**net_wm_icon_change**()

Called on `_NET_WM_ICON` change

X11 only. Called when a window notifies that it has changed its icon.

Arguments

- `Window` of client with changed icon

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.net_wm_icon_change
def icon_change(client):
    send_notification("qtile", f"{client.name} has changed its icon")
```

subscribe.**restart**()

Called before qtile is restarted.

This hook fires before qtile restarts but after qtile has checked that it is able to restart (i.e. the config file is valid).

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.restart
def run_every_startup():
    send_notification("qtile", "Restarting...")
```

subscribe.**resume**()

Called when system wakes up from sleep, suspend or hibernate.

Relies on systemd's inhibitor dbus interface, via the dbus-next package.

Note: the hook is not fired when resuming from shutdown/reboot events. Use the "startup" hooks for those scenarios.

Arguments

None

subscribe.**screen_change**()

Called when the output configuration is changed (e.g. via randr in X11).

Note: If you have `reconfigure_screens = True` in your config then qtile will automatically reconfigure your screens when it detects a change to the screen configuration. This hook is fired *before* that reconfiguration takes place. The `screens_reconfigured` hook should be used where you want to trigger an event after the reconfiguration.

Arguments

- `xproto.randr.ScreenChangeNotify` event (X11) or None (Wayland).

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.screen_change
def screen_change(event):
    send_notification("qtile", "Screen change detected.")
```

subscribe.**screens_reconfigured**()

Called once `qtile.reconfigure_screens` has completed (e.g. if `reconfigure_screens` is set to `True` in your config).

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.screens_reconfigured
def screen_reconf():
    send_notification("qtile", "Screens have been reconfigured.")
```

subscribe.**selection_change**()

Called on selection change

X11 only. Fired when a selection property is changed (e.g. new selection created or existing selection is emptied)

Arguments

- name of the selection

- dictionary describing selection, containing `owner` and `selection` as keys

The selection owner will typically be `"PRIMARY"` when contents is highlighted and `"CLIPBOARD"` when contents is actively copied to the clipboard, e.g. with Ctrl + C.

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.selection_change
def selection_change(name, selection):
    send_notification(
        "qtile",
        f"Window {selection['owner']} has changed the {name} selection."
    )
```

`subscribe.`**`selection_notify`**`()`

Called on selection notify

X11 only. Fired when a selection is made in a window.

Arguments

- name of the selection
- dictionary describing selection, containing `owner` and `selection` as keys

The selection owner will typically be `"PRIMARY"` when contents is highlighted and `"CLIPBOARD"` when contents is actively copied to the clipboard, e.g. with Ctrl + C.

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.selection_notify
def selection_notify(name, selection):
    send_notification(
        "qtile",
        f"Window {selection['owner']} has made a selection in the {name} selection."
    )
```

`subscribe.`**`setgroup`**`()`

Called when group is put on screen.

This hook is fired in 3 situations: 1) When the screen changes to a new group 2) When two groups are switched 3) When a screen is focused

Arguments

None

Example:

```
from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.setgroup
```

(continues on next page)

(continued from previous page)

```
def setgroup():
    send_notification("qtile", "Group set")
```

subscribe.**shutdown**()

Called before qtile is shutdown.

Using a long-running command in this function will cause the shutdown to be delayed.

This hook is only fired when qtile is shutting down, if you want a command to be run when the system sleeps then you should use the `suspend` hook instead.

Arguments

None

Example:

```
import os
import subprocess

from libqtile import hook

@hook.subscribe.shutdown
def autostart:
    script = os.path.expanduser("~/.config/qtile/shutdown.sh")
    subprocess.run([script])
```

subscribe.**startup**()

Called when qtile is started. Unlike `startup_once`, this hook is fired on every start, including restarts.

When restarting, this hook is fired after qtile has restarted but before qtile tries to restore the session to the same state that it was in before the restart.

Arguments

None

Example:

```
import subprocess

from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.startup
def run_every_startup():
    send_notification("qtile", "Startup")
```

subscribe.**startup_complete**()

Called when qtile is started after all resources initialized.

This is the same as `startup` with the only difference being that this hook is fired after the saved state has been restored.

Arguments

None

Example:

```
import subprocess

from libqtile import hook
from libqtile.utils import send_notification

@hook.subscribe.startup_complete
def run_every_startup():
    send_notification("qtile", "Startup complete")
```

subscribe.**startup_once**()

Called when Qtile has started on first start

This hook is called exactly once per session (i.e. not on each `lazy.restart()`).

Arguments

None

Example:

```
import os
import subprocess

from libqtile import hook

@hook.subscribe.startup_once
def autostart():
    script = os.path.expanduser("~/.config/qtile/autostart.sh")
    subprocess.run([script])
```

subscribe.**suspend**()

Called when system is about to sleep, suspend or hibernate.

Relies on systemd's inhibitor dbus interface, via the dbus-next package.

When this hook is used, qtile will set an inhibitor that prevent the system from sleeping. The inhibitor is removed as soon as your function exits. You should therefore not use long-running code in this function.

Please note, this inhibitor will also only delay, not block, the computer's ability to sleep. The default delay is 5 seconds. If your function has not completed within that time, the machine will still sleep (see important note below).

You can increase this delay by setting `InhibitDelayMaxSec` in `logind.conf`. see: https://www.freedesktop.org/software/systemd/man/logind.conf.html

In addition, closing a laptop lid will ignore inhibitors by default. You can override this by setting `LidSwitchIgnoreInhibited=no` in `/etc/systemd/logind.conf`.

Important: The logind service creates an inhibitor by passing a reference to a lock file which must be closed to release the lock. Additional references to the lock may be created if you spawn processes with the `subprocess` module and these processes are running when the machine tries to suspend. As a result, it is strongly recommended that you launch any processes with `qtile.spawn(...)` as this will not create additional copies of the lock.

Arguments

None

Example:

```
from libqtile import hook, qtile

@hook.subscribe.suspend
def lock_on_sleep():
    # Run screen locker
    qtile.spawn("/path/to/screen_locker")
```

subscribe.**user**()

Use to create user-defined hooks.

The purpose of these hooks is to allow a hook to be fired by an external application.

Hooked functions can receive arguments but it is up to the application firing the hook to ensure the correct arguments are passed. No checking will be performed by qtile.

Example:

```
from libqtile import hook
from libqtile.log_utils import logger

@hook.subscribe.user("my_custom_hook")
def hooked_function():
  logger.warning("Custom hook received.")
```

The external script can then call the hook with the following command:

```
qtile cmd-obj -o cmd -f fire_user_hook -a my_custom_hook
```

Note: If the script will be run by a different user then you will need to pass the path to the socket file used by the current process. One way to achieve this is to specify a path for the socket when starting qtile e.g. `qtile start -s /tmp/qtile.socket`. When firing the hook, you should then call `qtile cmd-obj -o cmd -f fire_user_hook -a my_custom_hook -s /tmp/qtile.socket` However, the same socket will need to be passed wherever you run `qtile cmd-obj` or `qtile shell`.

CHAPTER

TEN

BUILT-IN EXTENSIONS

10.1 CommandSet

class libqtile.extension.**CommandSet**(***config*)

Give list of commands to be executed in dmenu style.

ex. manage mocp deamon:

```
Key([mod], 'm', lazy.run_extension(extension.CommandSet(
    commands={
        'play/pause': '[ $(mocp -i | wc -l) -lt 2 ] && mocp -p || mocp -G',
        'next': 'mocp -f',
        'previous': 'mocp -r',
        'quit': 'mocp -x',
        'open': 'urxvt -e mocp',
        'shuffle': 'mocp -t shuffle',
        'repeat': 'mocp -t repeat',
        },
    pre_commands=['[ $(mocp -i | wc -l) -lt 1 ] && mocp -S'],
    **Theme.dmenu))),
```

ex. CommandSet inside another CommandSet

```
CommandSet(
    commands={
        "Hello": CommandSet(
            commands={
                "World": "echo 'Hello, World!'"
            },
            **Theme.dmenu
        )
    },
**Theme.dmenu
)
```

Configuration options

key	default	description
background	None	defines the normal background color (#RGB or #RRGGBB)
command	None	the command to be launched (string or list with arguments)
commands	None	dictionary of commands where key is runable command
dmenu_bottom	False	dmenu appears at the bottom of the screen
dmenu_command	'dmenu'	the dmenu command to be launched
dmenu_font	None	override the default 'font' and 'fontsize' options for dmenu
dmenu_height	None	defines the height (only supported by some dmenu forks)
dmenu_ignorecase	False	dmenu matches menu items case insensitively
dmenu_lines	None	dmenu lists items vertically, with the given number of lines
dmenu_prompt	None	defines the prompt to be displayed to the left of the input field
font	'sans'	defines the font name to be used
fontsize	None	defines the font size to be used
foreground	None	defines the normal foreground color (#RGB or #RRGGBB)
pre_commands	None	list of commands to be executed before getting dmenu answer
selected_backgro	None	defines the selected background color (#RGB or #RRGGBB)
selected_foregro	None	defines the selected foreground color (#RGB or #RRGGBB)

10.2 Dmenu

class libqtile.extension.**Dmenu**(***config*)

Python wrapper for dmenu http://tools.suckless.org/dmenu/

Configuration options

key	default	description
background	None	defines the normal background color (#RGB or #RRGGBB)
command	None	the command to be launched (string or list with arguments)
dmenu_bottom	False	dmenu appears at the bottom of the screen
dmenu_command	'dmenu'	the dmenu command to be launched
dmenu_font	None	override the default 'font' and 'fontsize' options for dmenu
dmenu_height	None	defines the height (only supported by some dmenu forks)
dmenu_ignorecase	False	dmenu matches menu items case insensitively
dmenu_lines	None	dmenu lists items vertically, with the given number of lines
dmenu_prompt	None	defines the prompt to be displayed to the left of the input field
font	'sans'	defines the font name to be used
fontsize	None	defines the font size to be used
foreground	None	defines the normal foreground color (#RGB or #RRGGBB)
selected_backgro	None	defines the selected background color (#RGB or #RRGGBB)
selected_foregro	None	defines the selected foreground color (#RGB or #RRGGBB)

10.3 DmenuRun

class libqtile.extension.**DmenuRun**(***config*)

Special case to run applications.

config.py should have something like:

```
from libqtile import extension
keys = [
    Key(['mod4'], 'r', lazy.run_extension(extension.DmenuRun(
        dmenu_prompt=">",
        dmenu_font="Andika-8",
        background="#15181a",
        foreground="#00ff00",
        selected_background="#079822",
        selected_foreground="#fff",
        dmenu_height=24,  # Only supported by some dmenu forks
    ))),
]
```

Configuration options

key	default	description
background	None	defines the normal background color (#RGB or #RRGGBB)
command	None	the command to be launched (string or list with arguments)
dmenu_bottom	False	dmenu appears at the bottom of the screen
dmenu_command	'dmenu_run'	the dmenu command to be launched
dmenu_font	None	override the default 'font' and 'fontsize' options for dmenu
dmenu_height	None	defines the height (only supported by some dmenu forks)
dmenu_ignorecase	False	dmenu matches menu items case insensitively
dmenu_lines	None	dmenu lists items vertically, with the given number of lines
dmenu_prompt	None	defines the prompt to be displayed to the left of the input field
font	'sans'	defines the font name to be used
fontsize	None	defines the font size to be used
foreground	None	defines the normal foreground color (#RGB or #RRGGBB)
selected_backgrc	None	defines the selected background color (#RGB or #RRGGBB)
selected_foregrc	None	defines the selected foreground color (#RGB or #RRGGBB)

10.4 J4DmenuDesktop

class libqtile.extension.**J4DmenuDesktop**(***config*)

Python wrapper for j4-dmenu-desktop https://github.com/enkore/j4-dmenu-desktop

Configuration options

key	default	description
background	None	defines the normal background color (#RGB or #RRGGBB)
command	None	the command to be launched (string or list with arguments)
dmenu_bottom	False	dmenu appears at the bottom of the screen
dmenu_command	'dmenu'	the dmenu command to be launched
dmenu_font	None	override the default 'font' and 'fontsize' options for dmenu
dmenu_height	None	defines the height (only supported by some dmenu forks)
dmenu_ignorecase	False	dmenu matches menu items case insensitively
dmenu_lines	None	dmenu lists items vertically, with the given number of lines
dmenu_prompt	None	defines the prompt to be displayed to the left of the input field
font	'sans'	defines the font name to be used
fontsize	None	defines the font size to be used
foreground	None	defines the normal foreground color (#RGB or #RRGGBB)
j4dmenu_command	'j4-dmenu-deskto	the dmenu command to be launched
j4dmenu_display_	False	display binary name after each entry
j4dmenu_generic	True	include the generic name of desktop entries
j4dmenu_terminal	None	terminal emulator used to start terminal apps
j4dmenu_usage_lo	None	file used to sort items by usage frequency
j4dmenu_use_xdg_	False	read $XDG_CURRENT_DESKTOP to determine the desktop environment
selected_backgro	None	defines the selected background color (#RGB or #RRGGBB)
selected_foregro	None	defines the selected foreground color (#RGB or #RRGGBB)

10.5 RunCommand

class libqtile.extension.**RunCommand**(***config*)

Run an arbitrary command.

Mostly useful as a superclass for more specific extensions that need to interact with the qtile object.

Also consider simply using lazy.spawn() or writing a client.

Configuration options

key	default	description
background	None	defines the normal background color (#RGB or #RRGGBB)
command	None	the command to be launched (string or list with arguments)
font	'sans'	defines the font name to be used
fontsize	None	defines the font size to be used
foreground	None	defines the normal foreground color (#RGB or #RRGGBB)
selected_backgro	None	defines the selected background color (#RGB or #RRGGBB)
selected_foregro	None	defines the selected foreground color (#RGB or #RRGGBB)

10.6 WindowList

class libqtile.extension.**WindowList**(***config*)

Give vertical list of all open windows in dmenu. Switch to selected.

Configuration options

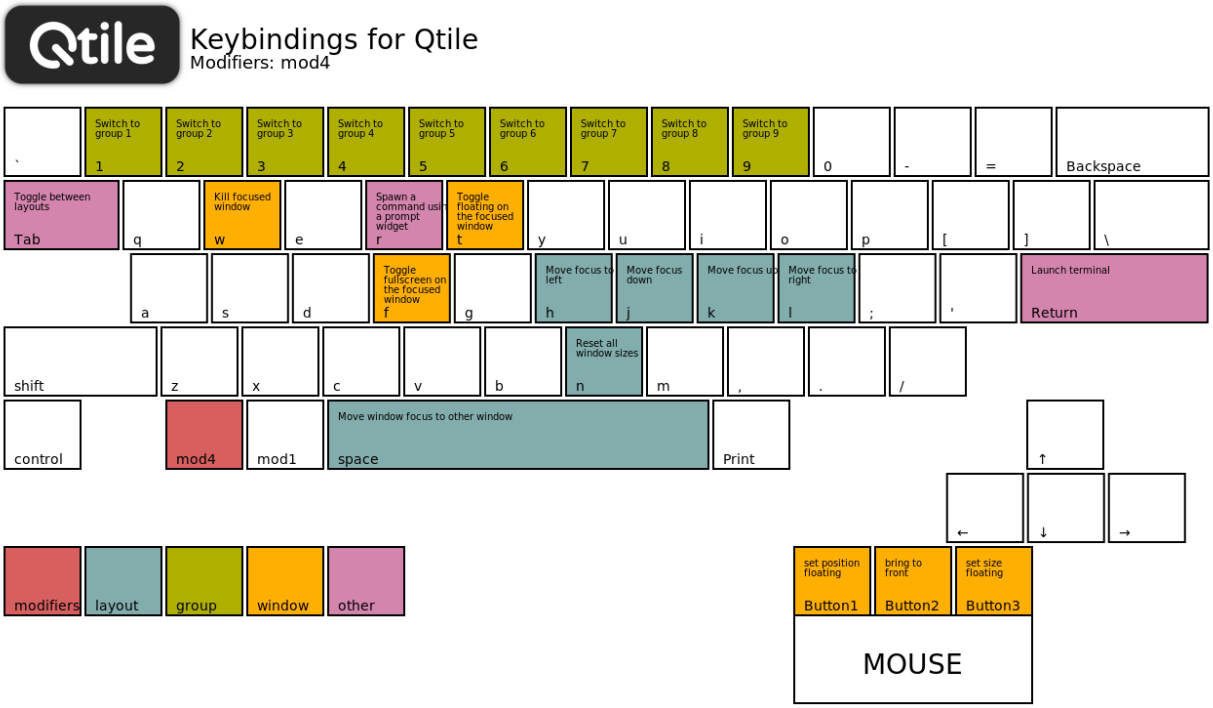
key	default	description
all_groups	True	If True, list windows from all groups; otherwise only from the current group
background	None	defines the normal background color (#RGB or #RRGGBB)
command	None	the command to be launched (string or list with arguments)
dmenu_bottom	False	dmenu appears at the bottom of the screen
dmenu_command	'dmenu'	the dmenu command to be launched
dmenu_font	None	override the default 'font' and 'fontsize' options for dmenu
dmenu_height	None	defines the height (only supported by some dmenu forks)
dmenu_ignorecase	False	dmenu matches menu items case insensitively
dmenu_lines	'80'	Give lines vertically. Set to None get inline
dmenu_prompt	None	defines the prompt to be displayed to the left of the input field
font	'sans'	defines the font name to be used
fontsize	None	defines the font size to be used
foreground	None	defines the normal foreground color (#RGB or #RRGGBB)
item_format	'{group}.{id}: {window}'	the format for the menu items
selected_backgro	None	defines the selected background color (#RGB or #RRGGBB)
selected_foregro	None	defines the selected foreground color (#RGB or #RRGGBB)

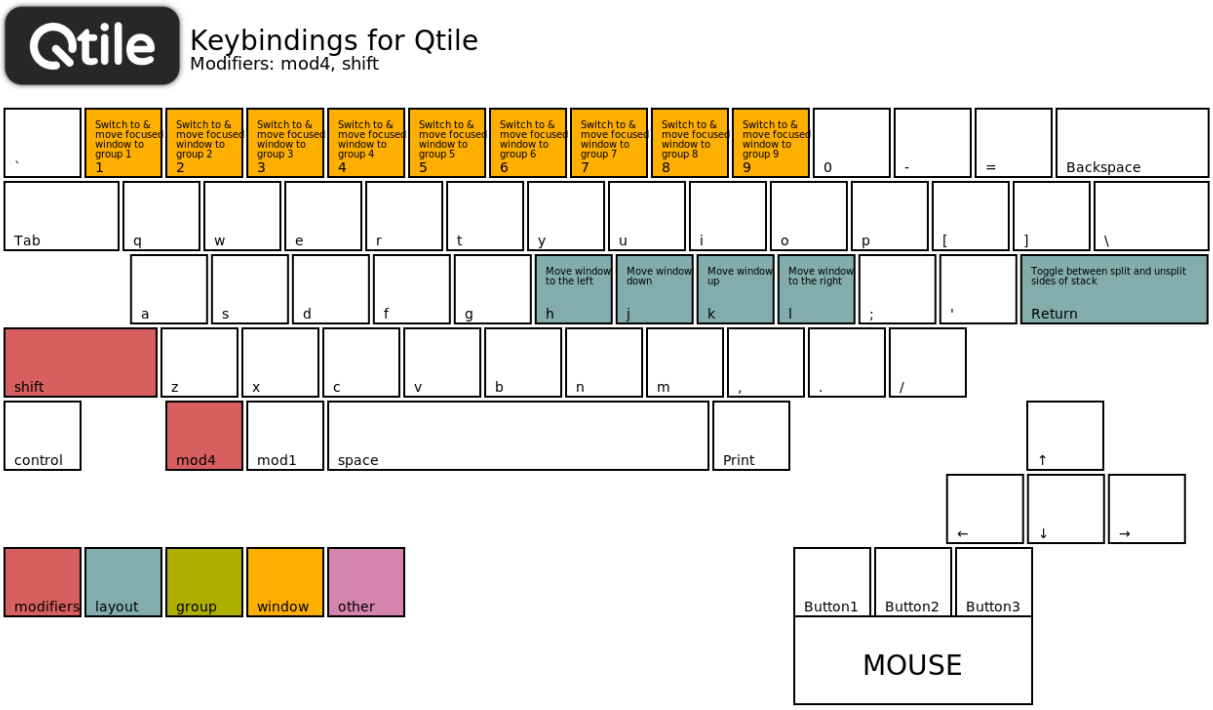
CHAPTER

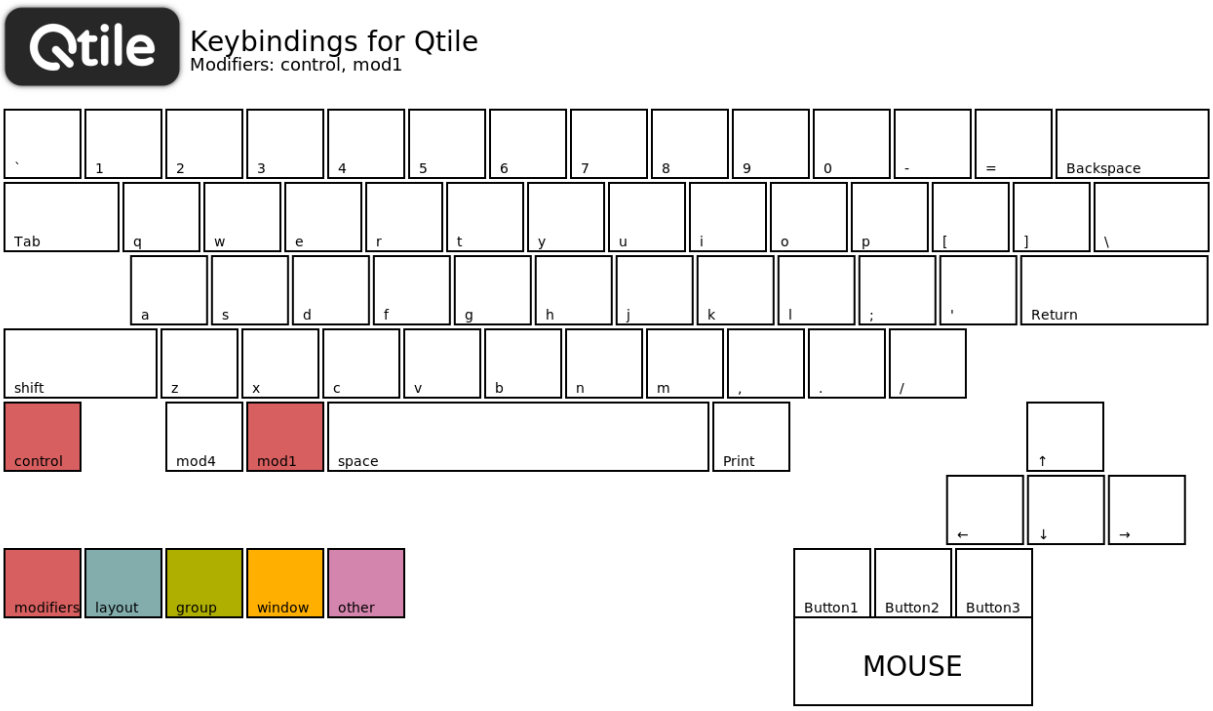
ELEVEN

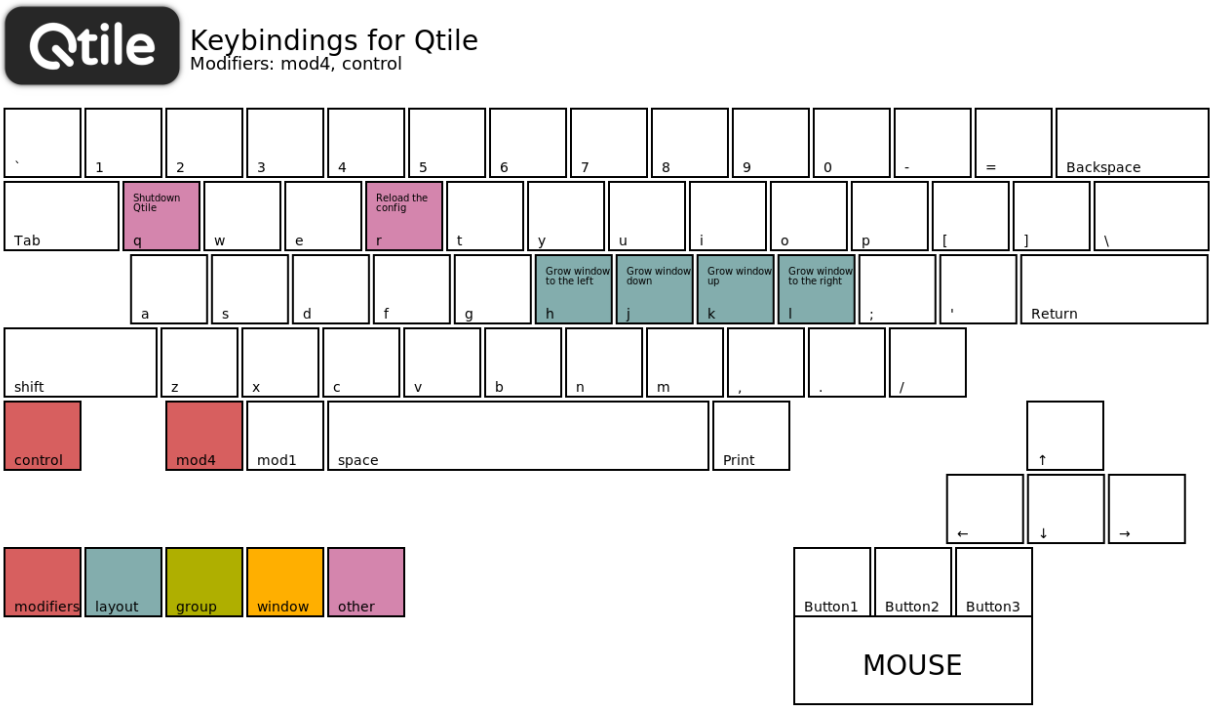
KEYBINDINGS IN IMAGES

11.1 Default configuration




Keybindings for Qtile
Modifiers: mod4, shift
` Switch to & move focused window to group 1 1 Switch to & move focused window to group 2 2 Switch to & move focused window to group 3 3 Switch to & move focused window to group 4 4 Switch to & move focused window to group 5 5 Switch to & move focused window to group 6 6 Switch to & move focused window to group 7 7 Switch to & move focused window to group 8 8 Switch to & move focused window to group 9 9 0 - = Backspace
Tab q w e r t y u i o p [ ] \
a s d f g Move window to the left h Move window down j Move window up k Move window to the right l ; ' Toggle between split and unsplit sides of stack Return
shift z x c v b n m , . /
control mod4 mod1 space Print
↑
← ↓ →
modifiers layout group window other
Button1 Button2 Button3
MOUSE


Keybindings for Qtile
Modifiers: control, mod1
` 1 2 3 4 5 6 7 8 9 0 - = Backspace
Tab q w e r t y u i o p [ ] \
a s d f g h j k l ; ' Return
shift z x c v b n m , . /
control mod4 mod1 space Print
↑
← ↓ →
modifiers layout group window other
Button1 Button2 Button3
MOUSE



11.2 Generate your own images

Qtile provides a tiny helper script to generate keybindings images from a config file. In the repository, the script is located under `scripts/gen-keybinding-img`.

This script accepts a configuration file and an output directory. If no argument is given, the default configuration will be used and files will be placed in same directory where the command has been run.

```
usage: gen-keybinding-img [-h] [-c CONFIGFILE] [-o OUTPUT_DIR]

Qtile keybindings image generator

optional arguments:
    -h, --help          show this help message and exit
    -c CONFIGFILE, --config CONFIGFILE
                        use specified configuration file. If no presented
                        default will be used
    -o OUTPUT_DIR, --output-dir OUTPUT_DIR
                        set directory to export all images to
```

CHAPTER

TWELVE

WINDOW STACKING

A number of window commands (`move_up/down()`, `bring_to_front()` etc.) relate to the stacking order of windows.

The aim of this page is to provide more details as to how stacking is implemented in Qtile.

Important: Currently, stacking is only implemented in the X11 background. Support will be added to the Wayland backend in future and this page will be updated accordingly.

12.1 Layer priority groups

We have tried to adhere to the EWMH specification. Windows are therefore stacked, from the bottom, according to the following priority rules:

- windows of type _NET_WM_TYPE_DESKTOP
- windows having state _NET_WM_STATE_BELOW
- windows not belonging in any other layer
- windows of type _NET_WM_TYPE_DOCK (unless they have state _NET_WM_TYPE_BELOW) and windows having state _NET_WM_STATE_ABOVE
- focused windows having state _NET_WM_STATE_FULLSCREEN

Qtile had then added an additional layer so that `Scratchpad` windows are placed above everything else.

Tiled windows will open in the default, "windows not belonging in any other layer", layer. If `floats_kept_above` is set to `True` in the config then new floating windows will have the `_NET_WM_STATE_ABOVE` property set which will ensure they remain above tiled windows.

12.2 Moving windows

Imagine you have four tiled windows stacked (from the top) as follows:

```
"One"
"Two"
"Three"
"Four"
```

If you call `move_up()` on window "Four", the result will be:

```
"One"
"Two"
"Four"
"Three"
```

If you now call `move_to_top()` on window "Three", the result will be:

```
"Three"
"One"
"Two"
"Four"
```

Note: `bring_to_front()` has a special behaviour in Qtile. This will bring any window to the very top of the stack, disregarding the priority rules set out above. When that window loses focus, it will be restacked in the appropriate location.

This can cause undesirable results if the config contains `bring_front_click=True` and the user has an app like a dock which is activated by mousing over the window. In this situation, tiled windows will be displayed above the dock making it difficult to activate. To fix this, set `bring_front_click` to `False` to disable the behaviour completely, or `"floating_only"` to only have this behaviour apply to floating windows.

CHAPTER

THIRTEEN

ARCHITECTURE

This page explains how Qtile's API works and how it can be accessed. Users who just want to find a list of commands can jump to *the API commands page*.

Qtile's command API is based on a graph of objects, where each object has a set of associated commands, combined with a number of interfaces that are used to navigate the graph and execute associated commands.

This page gives an overview of the command graph and the various interfaces accessible by users. The documentation also contains details of all the commands that are exposed by objects on the graph.

Note: While users are able to access the internal python objects (e.g. via a `qtile` instance), this is not part of the "official" API. These objects and method are not currently included in the documentation but can be viewed by looking at the source code on github. Changes to commonly-used internal objects will be kept to a minimum.

The graph and object commands are used in a number of different places:

- Commands can be *bound to keys* in the Qtile configuration file using the `lazy` interface.
- Commands can be called from a script using one of the various *available interfaces* to interact with Qtile from Python or shell scripts.

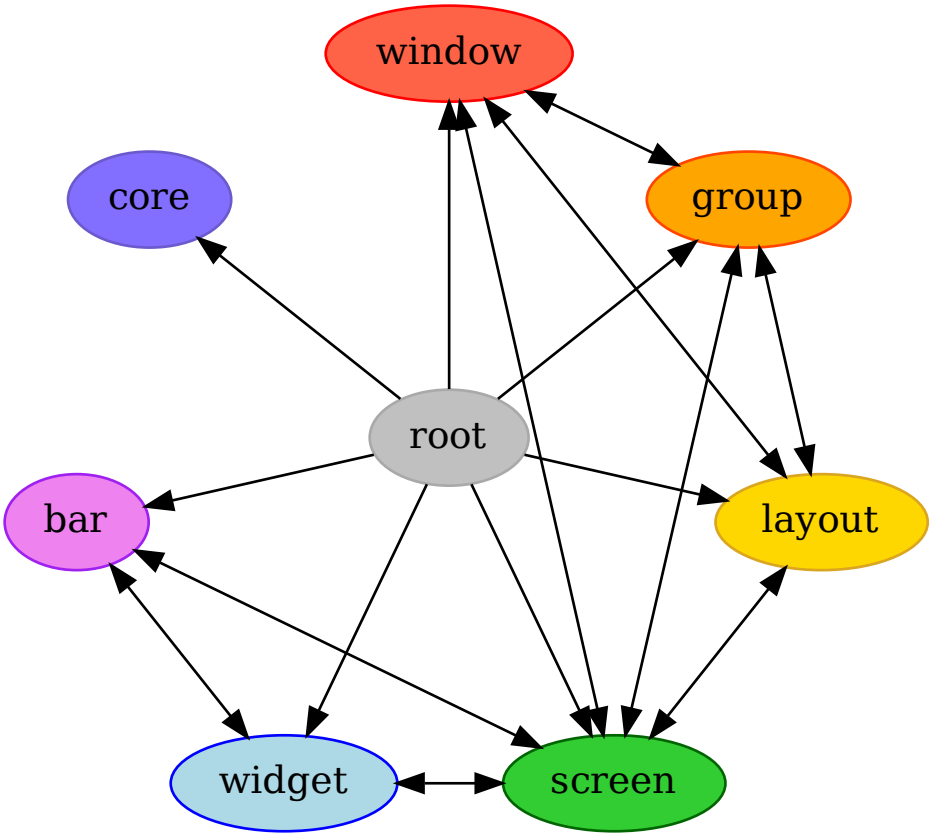
A couple of additional options are available if you are looking for more interactive access:

- Commands can be *called through qtile shell*, the Qtile shell.
- The shell can also be hooked into a Jupyter kernel *called iqshell* (NB this interface is currently broken).

If the explanations in the pages below seems a bit complex, please take a moment to explore the API using the `qtile shell` command shell. The shell provides a way to navigate the graph, allowing you to see how nodes are connected. Available nodes can be displayed with the `ls` command while command lists and detailed documentation can be accessed from the built-in `help` command. Commands can also be executed from this shell.

13.1 The Command Graph

The objects in Qtile's command graph come in eight flavours, matching the eight basic components of the window manager: `layouts`, `windows`, `groups`, `bars`, `widgets`, `screens`, `core`, and a special `root` node. Objects are addressed by a path specification that starts at the root and follows the available paths in the graph. This is what the graph looks like:



Each arrow can be read as "holds a reference to". So, we can see that a `widget` object *holds a reference to* objects of type `bar`, `screen` and `group`. Let's start with some simple examples of how the addressing works. Which particular objects we hold reference to depends on the context - for instance, widgets hold a reference to the screen that they appear on, and the bar they are attached to.

Let's look at an example, starting at the root node. The following script runs the `status` command on the root node, which, in this case, is represented by the `InteractiveCommandClient` object:

```
from libqtile.command.client import InteractiveCommandClient
c = InteractiveCommandClient()
print(c.status())
```

The `InteractiveCommandClient` is a class that allows us to traverse the command graph using attributes to select child nodes or commands. In this example, we have resolved the `status()` command on the root object. The interactive command client will automatically find and connect to a running Qtile instance, and which it will use to dispatch the call and print out the return.

An alternative is to use the `CommandClient`, which allows for a more precise resolution of command graph objects, but is not as easy to interact with from a REPL:

```
from libqtile.command.client import CommandClient
```

(continues on next page)

(continued from previous page)

```
c = CommandClient()
print(c.call("status")())
```

Like the interactive client, the command client will automatically connect to a running Qtile instance. Here, we first resolve the `status()` command with the `.call("status")`, which simply located the function, then we can invoke the call with no arguments.

For the rest of this example, we will use the interactive command client. From the graph, we can see that the root node holds a reference to `group` nodes. We can access the "info" command on the current group like so:

```
c.group.info()
```

To access a specific group, regardless of whether or not it is current, we use the Python mapping lookup syntax. This command sends group "b" to screen 1 (by the `libqtile.config.Group.toscreen()` method):

```
c.group["b"].toscreen(1)
```

In different contexts, it is possible to access a default object, where in other contexts a key is required. From the root of the graph, the current `group`, `layout`, `screen` and `window` can be accessed by simply leaving the key specifier out. The key specifier is mandatory for `widget` and `bar` nodes.

With this context, we can now drill down deeper in the graph, following the edges in the graphic above. To access the screen currently displaying group "b", we can do this:

```
c.group["b"].screen.info()
```

Be aware, however, that group "b" might not currently be displayed. In that case, it has no associated screen, the path resolves to a non-existent node, and we get an exception:

```
libqtile.command.CommandError: No object screen in path 'group['b'].screen'
```

The graph is not a tree, since it can contain cycles. This path (redundantly) specifies the group belonging to the screen that belongs to group "b":

```
c.group["b"].screen.group
```

This amount of connectivity makes it easy to reach out from a given object when callbacks and events fire on that object to related objects.

13.2 Navigating the command graph

As noted previously, some objects require a selector to ensure that the correct object is selected, while other nodes provide a default object without a selector.

The table below shows what selectors are required for the diferent nodes and whether the selector is optional (i.e. if it can be omitted to select the default object).

Object	Key	Optional?	Example
bar		No	
	"top", "bottom" (Note: if accessing this node from the root, users on multi-monitor setups may wish to navigate via a `screen` node to ensure that they select the correct object.)		c.screen.bar["bottom"]
group	Name string	Yes	
			c.group["one"] c.group
layout	Integer index	Yes	
			c.layout[2] c.layout
screen	Integer index	Yes	
			c.screen[1] c.screen
widget		No	
	Widget name (This is usually the name of the widget class in lower case but can be set by passing the `name` parameter to the widget.)		c.widget["textbox"]
window	Integer window ID	Yes	
			c.window[123456] c.window
core	No	n/a	
			c.core

13.3 Command graph development

This page provides further detail on how Qtile's command graph works. If you just want to script your Qtile window manager the *earlier information*, in addition to the documentation on the *available commands* should be enough to get started.

To develop the Qtile manager itself, we can dig into how Qtile represents these objects, which will lead to the way the commands are dispatched.

13.3.1 Client-Server Scripting Model

Qtile has a client-server control model - the main Qtile instance listens on a named pipe, over which marshalled command calls and response data is passed. This allows Qtile to be controlled fully from external scripts. Remote interaction occurs through an instance of the `libqtile.command.interface.IPCCommandInterface` class. This class establishes a connection to the currently running instance of Qtile. A `libqtile.command.client.InteractiveCommandClient` can use this connection to dispatch commands to the running instance. Commands then appear as methods with the appropriate signature on the `InteractiveCommandClient` object. The object hierarchy is described in the *Architecture* section of this manual. Full command documentation is available through the *Qtile Shell*.

13.3.2 Digging Deeper: Command Objects

All of the configured objects setup by Qtile are `CommandObject` subclasses. These objects are so named because we can issue commands against them using the command scripting API. Looking through the code, the commands that are exposed are commands that are decorated with the `@expose_command()` decorator. When writing custom layouts, widgets, or any other object, you can add your own custom functions and, once you add the decorator, they will be callable using the standard command infrastructure. An available command can be extracted by calling `.command()` with the name of the command.

In addition to having a set of associated commands, each command object also has a collection of items associated with it. This is what forms the graph that is shown above. For a given object type, the `items()` method returns all of the names of the associated objects of that type and whether or not there is a defaultable value. For example, from the root, `.items("group")` returns the name of all of the groups and that there is a default value, the currently focused group.

To navigate from one command object to the next, the `.select()` method is used. This method resolves a requested object from the command graph by iteratively selecting objects. A selector like `[("group", "b"), ("screen", None)]` would be to first resolve group "b", then the screen associated to the group.

13.3.3 The Command Graph

In order to help in specifying command objects, there is the abstract command graph structure. The command graph structure allows us to address any valid command object and issue any command against it without needing to have any Qtile instance running or have anything to resolve the objects to. This is particularly useful when constructing lazy calls, where the Qtile instance does not exist to specify the path that will be resolved when the command is executed. The only limitation of traversing the command graph is that it must follow the allowed edges specified in the first section above.

Every object in the command graph is represented by a `CommandGraphNode`. Any call can be resolved from a given node. In addition, each node knows about all of the children objects that can be reached from it and have the ability to `.navigate()` to the other nodes in the command graph. Each of the object types are represented as `CommandGraphObject` types and the root node of the graph, the `CommandGraphRoot` represents the Qtile instance.

When a call is performed on an object, it returns a `CommandGraphCall`. Each call will know its own name as well as be able to resolve the path through the command graph to be able to find itself.

Note that the command graph itself can standalone, there is no other functionality within Qtile that it relies on. While we could have started here and built up, it is helpful to understand the objects that the graph is meant to represent, as the graph is just a representation of a traversal of the real objects in a running Qtile window manager. In order to tie the running Qtile instance to the abstract command graph, we move on to the command interface.

13.3.4 Executing graph commands: Command Interface

The `CommandInterface` is what lets us take an abstract call on the command graph and resolve it against a running command object. Put another way, this is what takes the graph traversal `.group["b"].screen.info()` and executes the `info()` command against the addressed `screen` object. Additional functionality can be used to check that a given traversal resolves to actual objcets and that the requested command actually exists. Note that by construction of the command graph, the traversals here must be feasible, even if they cannot be resolved for a given configuration state. For example, it is possible to check the screen assoctiated to a group, even though the group may not be on a screen, but it is not possible to check the widget associated to a group.

The simplest form of the command interface is the `QtileCommandInterface`, which can take an in-process `Qtile` instance as the root `CommandObject` and execute requested commands. This is typically how we run the unit tests for Qtile.

The other primary example of this is the `IPCCommandInterface` which is able to then route all calls through an IPC client connected to a running Qtile instance. In this case, the command graph call can be constructed on the client side without having to dispatch to Qtile and once the call is constructed and deemed valid, the call can be executed.

In both of these cases, executing a command on a command interface will return the result of executing the command on a running Qtile instance. To support lazy execution, the `LazyCommandInterface` instead returns a `LazyCall` which is able to be resolved later by the running Qtile instance when it is configured to fire.

13.3.5 Tying it together: Command Client

So far, we have our running Command Objects and the Command Interface to dispatch commands against these objects as well as the Command Graph structure itself which encodes how to traverse the connections between the objects. The final component which ties everything together is the Command Client, which allows us to navigate through the graph to resolve objects, find their associated commands, and execute the commands against the held command interface.

The idea of the command client is that it is created with a reference into the command graph and a command interface. All navigation can be done against the command graph, and traversal is done by creating a new command client starting from the new node. When a command is executed against a node, that command is dispatched to the held command interface. The key decision here is how to perform the traversal. The command client exists in two different flavors: the standard `CommandClient` which is useful for handling more programatic traversal of the graph, calling methods to traverse the graph, and the `InteractiveCommandClient` which behaves more like a standard Python object, traversing by accessing properties and performing key lookups.

Returning to our examples above, we now have the full context to see what is going on when we call:

```
from libqtile.command.client import CommandClient
c = CommandClient()
print(c.call("status")())
from libqtile.command.client import InteractiveCommandClient
c = InteractiveCommandClient()
print(c.status())
```

In both cases, the command clients are constructed with the default command interface, which sets up an IPC connection to the running Qtile instance, and starts the client at the graph root. When we call `c.call("status")` or `c.status`, we navigate the command client to the `status` command on the root graph object. When these are invoked, the commands graph calls are dispatched via the IPC command interface and the results then sent back and printed on the local command line.

The power that can be realized by separating out the traversal and resolution of objects in the command graph from actually invoking or looking up any objects within the graph can be seen in the `lazy` module. By creating a lazy evaluated command client, we can expose the graph traversal and object resolution functionality via the same `InteractiveCommandClient` that is used to perform live command execution in the Qtile prompt.

CHAPTER

FOURTEEN

INTERFACES

14.1 Introduction

This page provides an overview of the various interfaces available to interact with Qtile's command graph.

- `lazy` calls
- when running `qtile shell`
- when running `qtile cmd-obj`
- when using `CommandClient` or `InteractiveCommandClient` in python

The way that these commands are called varies depending on which option you select. However, all interfaces follow the same, basic approach: navigate to the desired object and then execute a command on that object. The following examples illustrate this principle by showing how the same command can be accessed by the various interfaces:

```
Lazy call:
lazy.widget["volume"].increase_volume()

qtile shell:
> cd widget/volume
widget[volume] > increase_volume()

qtile cmd-obj:
qtile cmd-obj -o widget volume -f increase_volume

CommandClient:
>>> from libqtile.command.client import CommandClient
>>> c = CommandClient()
>>> c.navigate("widget", "volume").call("increase_volume")

InteractiveCommandClient:
>>> from libqtile.command.client import InteractiveCommandClient
>>> c = InteractiveCommandClient()
>>> c.widget["volume"].increase_volume()
```

14.2 The Interfaces

From the examples above, you can see that there are five main interfaces which can be used to interact with Qtile's command graph. Which one you choose will depend on how you intend to use it as each interface is suited to different scenarios.

- The `lazy` interface is used in config scripts to bind commands to keys and mouse callbacks.
- The `qtile shell` is a tool for exploring the graph by presenting it as a file structure. It is not designed to be used for scripting.
- For users creating shell scripts, the `qtile cmd-obj` interface would be the recommended choice.
- For users wanting to control Qtile from a python script, there are two available interfaces `libqtile.command.client.CommandClient` and `libqtile.command.client.InteractiveCommandClient`. Users are advised to use the `InteractiveCommandClient` as this simplifies the syntax for navigating the graph and calling commands.

14.2.1 The Lazy interface

The `lazy.lazy` object is a special helper object to specify a command for later execution. Lazy objects are typically users' first exposure to Qtile's command graph but they may not realise it. However, understanding this will help users when they try using some of the other interfaces listed on this page.

The basic syntax for a lazy command is:

```
lazy.node[selector].command(arguments)
```

No node is required when accessing commands on the root node. In addition, multiple nodes can be sequenced if required to navigate to a specific object. For example, bind a key that would focus the next window on the active group on screen 2, you would create a lazy object as follows:

```
lazy.screen[1].group.next_window()
```

Note: As noted above, `lazy` calls do not call the relevant command but only create a reference to it. While this makes it ideal for binding commands to key presses and `mouse_callbacks` for widgets, it also means that `lazy` calls cannot be included in user-defined functions.

14.2.2 qtile shell

The qtile shell maps the command graph to a virtual filesystem that can be navigated in a similar way. While it is unlikely to be used for scripting, the `qtile shell` interface provides an excellent means for users to navigate and familiarise themselves with the command graph.

For more information, please refer to *qtile shell*

14.2.3 qtile cmd-obj

`qtile cmd-obj` is a command line interface for executing commands on the command graph. It can be used as a standalone command (e.g. executed directly from the terminal) or incorporated into shell scripts.

For more information, please refer to *qtile cmd-obj*

14.2.4 CommandClient

The `CommandClient` interface is a low-level python interface for accessing and navigating the command graph. The low-level nature means that navigation steps must be called explicitly, rather than being inferred from the body of the calling command.

For example:

```
from libqtile.command.client import CommandClient

c = CommandClient()

# Call info command on clock widget
info = c.navigate("widget", "clock").call("info")

# Call info command on the screen displaying the clock widget
info = c.navigate("widget", "clock").navigate("screen", None).call("info")
```

Note from the last example that each navigation step must be called separately. The arguments passed to `navigate()` are `node` and `selector`. `selector` is `None` when you wish to access the default object on that node (e.g. the current screen).

More technical explanation about the python command clients can be found at *Executing graph commands: Command Interface*.

14.2.5 InteractiveCommandClient

The `InteractiveCommandClient` is likely to be the more popular interface for users wishing to access the command graph via external python scripts. One of the key differences between the `InteractiveCommandClient` and the above `CommandClient` is that the `InteractiveCommandClient` removes the need to call `navigate` and `call` explicitly. Instead, the syntax mimics that of the `lazy` interface.

For example, to call the same commands in the above example:

```
from libqtile.command.client import InteractiveCommandClient

c = InteractiveCommandClient()

# Call info command on clock widget
info = c.widget["clock"].info()

# Call info command on the screen displaying the clock widget
info = c.widget["clock"].screen.info()
```

CHAPTER

FIFTEEN

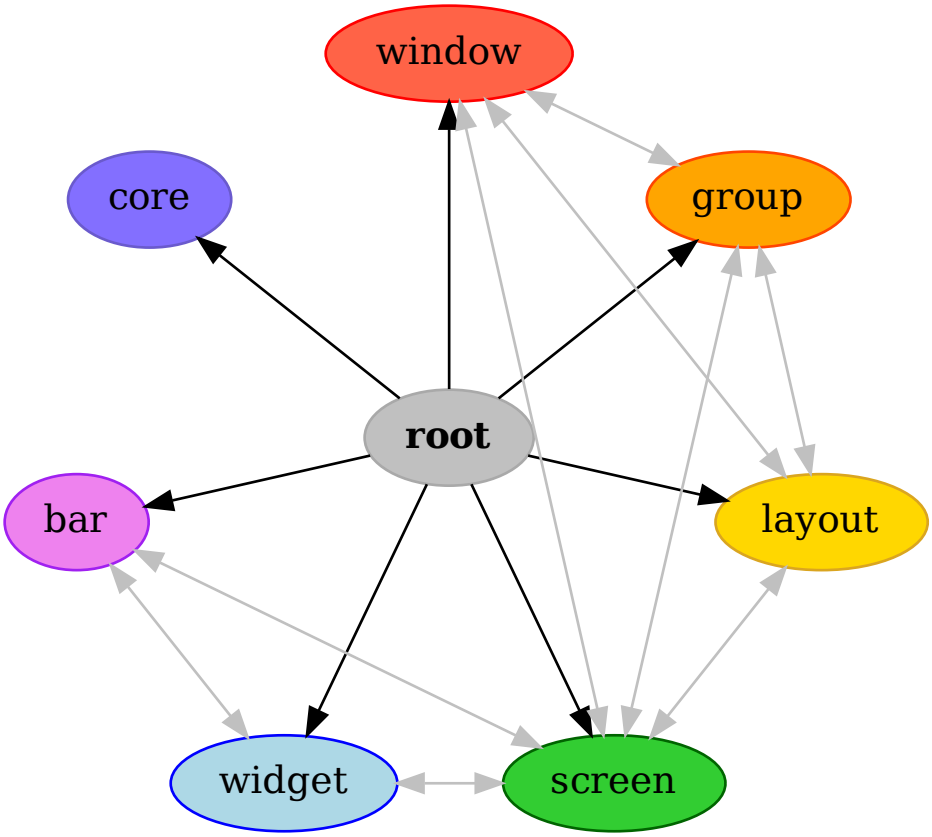
COMMANDS API

The following pages list all the commands that are exposed by Qtile's command graph. As a result, all of these commands are accessible by any of the various interfaces provided by Qtile (e.g. the `lazy` interface for keybindings and mouse callbacks).

15.1 Qtile root object

The root node represents the main Qtile manager instance. Many of the commands on this node are therefore related to the running of the application itself.

The root can access every other node in the command graph. Certain objects can be accessed without a selector resulting in the current object being selected (e.g. current group, screen, layout, window).



class libqtile.core.manager.**Qtile**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.<command>()
qtile cmd-obj -o cmd -f <command>

The following commands are available for this object:

add_rule(match_args, rule_args[, min_priorty])	Add a dgroup rule, returns rule_id needed to remove it
addgroup(group[, label, layout, layouts, index])	Add a group with the given name

continues on next page

Table 1 – continued from previous page

change_window_order(new_location)	Change the order of the current window within the current group.
commands()	Returns a list of possible commands for this object
critical()	Set log level to CRITICAL
debug()	Set log level to DEBUG
delgroup(group)	Delete a group with the given name
display_kb()	Display table of key bindings
doc(name)	Returns the documentation for a specified command name
error()	Set log level to ERROR
eval(code)	Evaluates code in the same context as this function
findwindow([prompt, widget])	Launch prompt widget to find a window of the given name
fire_user_hook(hook_name, *args)	Fire a custom hook.
function(function, *args, **kwargs)	Call a function with current object as argument
get_groups()	Return a dictionary containing information for all groups
get_screens()	Return a list of dictionaries providing information on all screens
get_state()	Get pickled state for restarting qtile
get_test_data()	Returns any content arbitrarily set in the self.test_data attribute.
hide_show_bar([position])	Toggle visibility of a given bar
info()	Set log level to INFO
internal_windows()	Return info for each internal window (bars, for example)
items(name)	Build a list of contained items for the given item class.
labelgroup([prompt, widget])	Launch prompt widget to label the current group
list_widgets()	List of all addressible widget names
loglevel()	
loglevelname()	
next_layout([name])	Switch to the next layout.
next_screen()	Move to next screen
next_urgent()	Focus next window with urgent hint
pause()	Drops into pdb
prev_layout([name])	Switch to the previous layout.
prev_screen()	Move to the previous screen
qtile_info()	Returns a dictionary of info on the Qtile instance
qtilecmd([prompt, widget, messenger])	Execute a Qtile command using the client syntax
reconfigure_screens(*_, **__)	This can be used to set up screens again during run time.
reload_config()	Reload the configuration file.
remove_rule(rule_id)	Remove a dgroup rule by rule_id
restart()	Restart Qtile.
run_extension(extension)	Run extensions
shutdown()	Quit Qtile
simulate_keypress(modifiers, key)	Simulates a keypress on the focused window.
spawn(cmd[, shell, env])	Spawn a new process.
spawncmd([prompt, widget, command, ...])	Spawn a command using a prompt widget, with tab-completion.

continues on next page

Table 1 – continued from previous page

status()	Return "OK" if Qtile is running
switch_groups(namea, nameb)	Switch position of two groups by name
switch_window(location)	Change to the window at the specified index in the current group.
switchgroup([prompt, widget])	Launch prompt widget to switch to a given group to the current screen
sync()	Sync the backend's event queue.
to_layout_index(index[, name])	Switch to the layout with the given index in self.layouts.
to_screen(n)	Warp focus to screen n, where n is a 0-based screen number
togroup([prompt, widget])	Launch prompt widget to move current window to a given group
tracemalloc_dump()	Dump tracemalloc snapshot
tracemalloc_toggle()	Toggle tracemalloc status
ungrab_all_chords()	Leave all chord modes and grab the root bindings
ungrab_chord()	Leave a chord mode
validate_config()	
warning()	Set log level to WARNING
windows()	Return info for each client window

Command documentation

add_rule(*match_args: dict[str, Any]*, *rule_args: dict[str, Any]*, *min_priorty: bool = False*) → int | None

Add a dgroup rule, returns rule_id needed to remove it

Parameters

match_args
config.Match arguments

rule_args
config.Rule arguments

min_priorty
If the rule is added with minimum priority (last) (default: False)

addgroup(*group: str*, *label: str | None = None*, *layout: str | None = None*, *layouts: list[Layout] | None = None*, *index: int | None = None*) → bool

Add a group with the given name

change_window_order(*new_location: int*) → None

Change the order of the current window within the current group.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

critical() → None

Set log level to CRITICAL

debug() → None

Set log level to DEBUG

delgroup(*group: str*) → None
Delete a group with the given name

display_kb() → str
Display table of key bindings

doc(*name*) → str
Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

error() → None
Set log level to ERROR

eval(*code: str*) → tuple[bool, str | None]
Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

findwindow(*prompt: str = 'window'*, *widget: str = 'prompt'*) → None
Launch prompt widget to find a window of the given name

Parameters

prompt
Text with which to prompt user (default: "window")

widget
Name of the prompt widget (default: "prompt")

fire_user_hook(*hook_name: str*, **args: Any*) → None
Fire a custom hook.

function(*function*, **args*, ***kwargs*) → None
Call a function with current object as argument

get_groups() → dict[str, dict[str, Any]]
Return a dictionary containing information for all groups

Examples

get_groups()

get_screens() → list[dict[str, Any]]
Return a list of dictionaries providing information on all screens

get_state() → str
Get pickled state for restarting qtile

get_test_data() → Any
Returns any content arbitrarily set in the self.test_data attribute. Useful in tests.

hide_show_bar(*position: Literal['top', 'bottom', 'left', 'right', 'all'] = 'all'*) → None
Toggle visibility of a given bar

Parameters

position
one of: "top", "bottom", "left", "right", or "all" (default: "all")

info() → None

Set log level to INFO

internal_windows() → list[dict[str, Any]]

Return info for each internal window (bars, for example)

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

labelgroup(*prompt: str = 'label'*, *widget: str = 'prompt'*) → None

Launch prompt widget to label the current group

Parameters

prompt
Text with which to prompt user (default: "label")

widget
Name of the prompt widget (default: "prompt")

list_widgets() → list[str]

List of all addressible widget names

loglevel() → int

loglevelname() → str

next_layout(*name: str | None = None*) → None

Switch to the next layout.

Parameters

name
Group name. If not specified, the current group is assumed

next_screen() → None

Move to next screen

next_urgent() → None

Focus next window with urgent hint

pause() → None

Drops into pdb

prev_layout(*name: str | None = None*) → None

Switch to the previous layout.

Parameters

name
Group name. If not specified, the current group is assumed

prev_screen() → None

Move to the previous screen

qtile_info() → dict

Returns a dictionary of info on the Qtile instance

qtilecmd(*prompt: str = 'command'*, *widget: str = 'prompt'*, *messenger: str = 'xmessage'*) → None

Execute a Qtile command using the client syntax

Tab completion aids navigation of the command tree

Parameters

prompt
Text to display at the prompt (default: "command: ")

widget
Name of the prompt widget (default: "prompt")

messenger
Command to display output, set this to None to disable (default: "xmessage")

reconfigure_screens(**_: list[Any]*, ***__: dict[Any, Any]*) → None

This can be used to set up screens again during run time. Intended usage is to be called when the screen_change hook is fired, responding to changes in physical monitor setup by configuring qtile.screens accordingly. The args are ignored; it is here in case this function is hooked directly to screen_change.

reload_config() → None

Reload the configuration file.

Can also be triggered by sending Qtile a SIGUSR1 signal.

remove_rule(*rule_id: int*) → None

Remove a dgroup rule by rule_id

restart() → None

Restart Qtile.

Can also be triggered by sending Qtile a SIGUSR2 signal.

run_extension(*extension: _Extension*) → None

Run extensions

shutdown() → None

Quit Qtile

simulate_keypress(*modifiers: list[str]*, *key: str*) → None

Simulates a keypress on the focused window.

This triggers internal bindings only; for full simulation see external tools such as xdotool or ydotool.

Parameters

modifiers
A list of modifier specification strings. Modifiers can be one of "shift", "lock", "control" and "mod1" - "mod5".

key
Key specification.

Examples

simulate_keypress(["control", "mod2"], "k")

spawn(*cmd: list[str] | str*, *shell: bool = False*, *env: dict[str, str] = {}*) → int

Spawn a new process.

Parameters

cmd:
The command to execute either as a single string or list of strings.

shell:
Whether to execute the command in a new shell by prepending it with "/bin/sh -c". This enables the use of shell syntax within the command (e.g. pipes).

env:
Dictionary of environmental variables to pass with command.

Examples

spawn("firefox")

spawn(["xterm", "-T", "Temporary terminal"])

spawn("screenshot | xclip", shell=True)

spawncmd(*prompt: str = 'spawn'*, *widget: str = 'prompt'*, *command: str = '%s'*, *complete: str = 'cmd'*, *shell: bool = True*, *aliases: dict[str, str] | None = None*) → None

Spawn a command using a prompt widget, with tab-completion.

Parameters

prompt
Text with which to prompt user (default: "spawn: ").

widget
Name of the prompt widget (default: "prompt").

command
command template (default: "%s").

complete
Tab completion function (default: "cmd")

shell
Execute the command with /bin/sh (default: True)

aliases
Dictionary mapping aliases to commands. If the entered command is a key in this dict, the command it maps to will be executed instead.

status() → Literal['OK']

Return "OK" if Qtile is running

switch_groups(*namea: str*, *nameb: str*) → None

Switch position of two groups by name

switch_window(*location: int*) → None

Change to the window at the specified index in the current group.

switchgroup(*prompt: str = 'group'*, *widget: str = 'prompt'*) → None

Launch prompt widget to switch to a given group to the current screen

Parameters

prompt
Text with which to prompt user (default: "group")

widget
Name of the prompt widget (default: "prompt")

sync() → None

Sync the backend's event queue. Should only be used for development.

to_layout_index(*index: int*, *name: str | None = None*) → None

Switch to the layout with the given index in self.layouts.

Parameters

index
Index of the layout in the list of layouts.

name
Group name. If not specified, the current group is assumed.

to_screen(*n: int*) → None

Warp focus to screen n, where n is a 0-based screen number

Examples

to_screen(0)

togroup(*prompt: str = 'group'*, *widget: str = 'prompt'*) → None

Launch prompt widget to move current window to a given group

Parameters

prompt
Text with which to prompt user (default: "group")

widget
Name of the prompt widget (default: "prompt")

tracemalloc_dump() → tuple[bool, str]

Dump tracemalloc snapshot

tracemalloc_toggle() → None

Toggle tracemalloc status

Running tracemalloc is required for *qtile top*

ungrab_all_chords() → None

Leave all chord modes and grab the root bindings

ungrab_chord() → None

Leave a chord mode

validate_config() → None

warning() → None

Set log level to WARNING

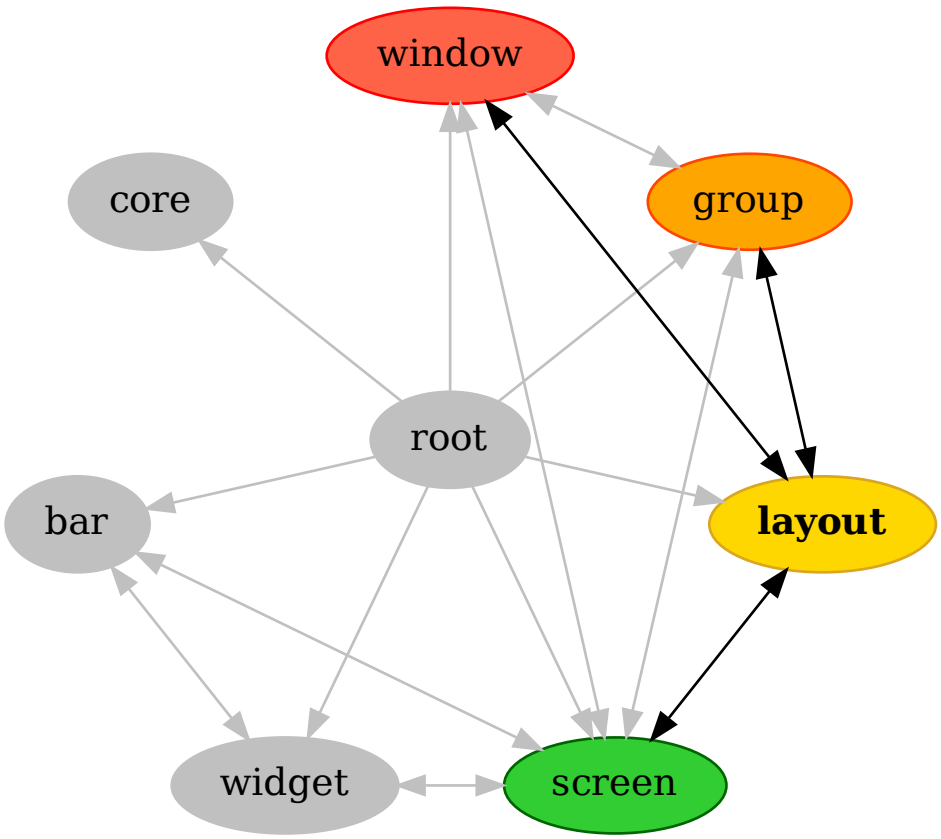
windows() → list[dict[str, Any]]

Return info for each client window

15.2 Layout objects

Layouts position windows according to their specific rules. Layout commands typically include moving windows around the layout and changing the size of windows.

Layouts can access the windows being displayed, the group holding the layout and the screen displaying the layout.



15.2.1 Bsp

class libqtile.layout.bsp.**Bsp**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
down()	
eval(code)	Evaluates code in the same context as this function
flip_down()	
flip_left()	
flip_right()	
flip_up()	
function(function, *args, **kwargs)	Call a function with current object as argument
grow_down()	
grow_left()	
grow_right()	
grow_up()	
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
left()	
next()	
normalize()	
previous()	
right()	
shuffle_down()	
shuffle_left()	
shuffle_right()	
shuffle_up()	
toggle_split()	
up()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

down()

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

flip_down()

flip_left()

flip_right()

flip_up()

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

grow_down()

grow_left()

grow_right()

grow_up()

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

left()

next() → None

normalize()

previous() → None

right()

shuffle_down()

shuffle_left()

shuffle_right()

shuffle_up()

toggle_split()

up()

15.2.2 Columns

class libqtile.layout.columns.**Columns**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
down()	
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
grow_down()	
grow_left()	
grow_right()	
grow_up()	
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
left()	
next()	
normalize()	
previous()	
right()	
shuffle_down()	
shuffle_left()	
shuffle_right()	
shuffle_up()	
swap_column_left()	
swap_column_right()	
toggle_split()	
up()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

down()

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

grow_down()

grow_left()

grow_right()

grow_up()

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

left()

next() → None

normalize()

previous() → None

right()

shuffle_down()

shuffle_left()

shuffle_right()

shuffle_up()

swap_column_left()

swap_column_right()

toggle_split()

up()

15.2.3 Floating

class libqtile.layout.floating.**Floating**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
next()	
previous()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next() → None

previous() → None

15.2.4 Matrix

class libqtile.layout.matrix.**Matrix**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

add()	Increase number of columns
commands()	Returns a list of possible commands for this object
delete()	Decrease number of columns
doc(name)	Returns the documentation for a specified command name
down()	Switch to the next window in current column
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
left()	Switch to the next window on current row
next()	
previous()	
right()	Switch to the next window on current row
up()	Switch to the previous window in current column

Command documentation

add()

Increase number of columns

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

delete()

Decrease number of columns

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

down()

Switch to the next window in current column

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

left()

Switch to the next window on current row

next() → None

previous() → None

right()

Switch to the next window on current row

up()

Switch to the previous window in current column

15.2.5 Max

class libqtile.layout.max.**Max**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
down()	
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
next()	
previous()	
up()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

down()

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next()

previous()

up()

15.2.6 MonadTall

class libqtile.layout.xmonad.**MonadTall**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
flip()	Flip the layout horizontally
function(function, *args, **kwargs)	Call a function with current object as argument
grow()	Grow current window
grow_main()	Grow main pane
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
left()	Focus on the closest window to the left of the current window
maximize()	Grow the currently focused client to the max size
next()	
normalize([redraw])	Evenly distribute screen-space among secondary clients
previous()	
reset([ratio, redraw])	Reset Layout.
right()	Focus on the closest window to the right of the current window
set_ratio(ratio)	Directly set the main pane ratio
shrink()	Shrink current window
shrink_main()	Shrink main pane
shuffle_down()	Shuffle the client down the stack
shuffle_up()	Shuffle the client up the stack
swap(window1, window2)	Swap two windows
swap_left()	Swap current window with closest window to the left
swap_main()	Swap current window to main pane
swap_right()	Swap current window with closest window to the right

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

flip()

Flip the layout horizontally

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

grow()

Grow current window

Will grow the currently focused client reducing the size of those around it. Growing will stop when no other secondary clients can reduce their size any further.

grow_main()

Grow main pane

Will grow the main pane, reducing the size of clients in the secondary pane.

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

left()

Focus on the closest window to the left of the current window

maximize()

Grow the currently focused client to the max size

next() → None

normalize(*redraw=True*)

Evenly distribute screen-space among secondary clients

previous() → None

reset(*ratio=None*, *redraw=True*)

Reset Layout.

right()

Focus on the closest window to the right of the current window

set_ratio(*ratio*)

Directly set the main pane ratio

shrink()

Shrink current window

Will shrink the currently focused client reducing the size of those around it. Shrinking will stop when the client has reached the minimum size.

shrink_main()

Shrink main pane

Will shrink the main pane, increasing the size of clients in the secondary pane.

shuffle_down()

Shuffle the client down the stack

shuffle_up()

Shuffle the client up the stack

swap(*window1:* Window, *window2:* Window) → None

Swap two windows

swap_left()

Swap current window with closest window to the left

swap_main()

Swap current window to main pane

swap_right()

Swap current window with closest window to the right

15.2.7 MonadThreeCol

class libqtile.layout.xmonad.**MonadThreeCol**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
flip()	Flip the layout horizontally
function(function, *args, **kwargs)	Call a function with current object as argument
grow()	Grow current window
grow_main()	Grow main pane
items(name)	Build a list of contained items for the given item class.
left()	Focus on the closest window to the left of the current window
maximize()	Grow the currently focused client to the max size
next()	
normalize([redraw])	Evenly distribute screen-space among secondary clients
previous()	
reset([ratio, redraw])	Reset Layout.
right()	Focus on the closest window to the right of the current window
set_ratio(ratio)	Directly set the main pane ratio
shrink()	Shrink current window
shrink_main()	Shrink main pane
shuffle_down()	Shuffle the client down the stack
shuffle_up()	Shuffle the client up the stack
swap(window1, window2)	Swap two windows
swap_left()	Swap current window with closest window to the left
swap_main()	Swap current window to main pane
swap_right()	Swap current window with closest window to the right

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

flip()

Flip the layout horizontally

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

grow()

Grow current window

Will grow the currently focused client reducing the size of those around it. Growing will stop when no other secondary clients can reduce their size any further.

grow_main()

Grow main pane

Will grow the main pane, reducing the size of clients in the secondary pane.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

left()

Focus on the closest window to the left of the current window

maximize()

Grow the currently focused client to the max size

next() → None

normalize(*redraw=True*)

Evenly distribute screen-space among secondary clients

previous() → None

reset(*ratio=None*, *redraw=True*)

Reset Layout.

right()

Focus on the closest window to the right of the current window

set_ratio(*ratio*)

Directly set the main pane ratio

shrink()

Shrink current window

Will shrink the currently focused client reducing the size of those around it. Shrinking will stop when the client has reached the minimum size.

shrink_main()

Shrink main pane

Will shrink the main pane, increasing the size of clients in the secondary pane.

shuffle_down()

Shuffle the client down the stack

shuffle_up()

Shuffle the client up the stack

swap(*window1:* Window, *window2:* Window) → None

Swap two windows

swap_left()

Swap current window with closest window to the left

swap_main()

Swap current window to main pane

swap_right()

Swap current window with closest window to the right

15.2.8 MonadWide

class libqtile.layout.xmonad.**MonadWide**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
flip()	Flip the layout horizontally
function(function, *args, **kwargs)	Call a function with current object as argument
grow()	Grow current window
grow_main()	Grow main pane
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
left()	Focus on the closest window to the left of the current window
maximize()	Grow the currently focused client to the max size
next()	
normalize([redraw])	Evenly distribute screen-space among secondary clients
previous()	
reset([ratio, redraw])	Reset Layout.
right()	Focus on the closest window to the right of the current window
set_ratio(ratio)	Directly set the main pane ratio
shrink()	Shrink current window
shrink_main()	Shrink main pane
shuffle_down()	Shuffle the client down the stack
shuffle_up()	Shuffle the client up the stack
swap(window1, window2)	Swap two windows
swap_left()	Swap current window with closest window to the down
swap_main()	Swap current window to main pane
swap_right()	Swap current window with closest window to the up

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

flip()

Flip the layout horizontally

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

grow()

Grow current window

Will grow the currently focused client reducing the size of those around it. Growing will stop when no other secondary clients can reduce their size any further.

grow_main()

Grow main pane

Will grow the main pane, reducing the size of clients in the secondary pane.

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

left()

Focus on the closest window to the left of the current window

maximize()

Grow the currently focused client to the max size

next() → None

normalize(*redraw=True*)

Evenly distribute screen-space among secondary clients

previous() → None

reset(*ratio=None*, *redraw=True*)

Reset Layout.

right()

Focus on the closest window to the right of the current window

set_ratio(*ratio*)

Directly set the main pane ratio

shrink()

Shrink current window

Will shrink the currently focused client reducing the size of those around it. Shrinking will stop when the client has reached the minimum size.

shrink_main()

Shrink main pane

Will shrink the main pane, increasing the size of clients in the secondary pane.

shuffle_down()

Shuffle the client down the stack

shuffle_up()

Shuffle the client up the stack

swap(*window1:* Window, *window2:* Window) → None

Swap two windows

swap_left()

Swap current window with closest window to the down

swap_main()

Swap current window to main pane

swap_right()

Swap current window with closest window to the up

15.2.9 RatioTile

class libqtile.layout.ratiotile.**RatioTile**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
decrease_ratio()	
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
increase_ratio()	
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
next()	
previous()	
shuffle_down()	
shuffle_up()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_ratio()

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

increase_ratio()

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next() → None

previous() → None

shuffle_down()

shuffle_up()

15.2.10 ScreenSplit

class libqtile.layout.screensplit.**ScreenSplit**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
move_window_to_next_split()	Move current window to next split.
move_window_to_previous_split()	Move current window to previous split.
next()	Move to next client.
next_split()	Move to next split.
previous()	Move to previous client.
previous_split()	Move to previous client.

Command documentation

commands()

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

move_window_to_next_split() → None

Move current window to next split.

move_window_to_previous_split() → None

Move current window to previous split.

next() → None

Move to next client.

next_split() → None

Move to next split.

previous() → None

Move to previous client.

previous_split() → None

Move to previous client.

15.2.11 Slice

class libqtile.layout.slice.**Slice**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
move_to_slice()	Moves the current window to the slice.
next()	
previous()	

Command documentation

commands()

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

move_to_slice()

Moves the current window to the slice.

next() → None

previous() → None

15.2.12 Spiral

class libqtile.layout.spiral.**Spiral**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
decrease_ratio()	Decrease spiral ratio.
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
grow_main()	Grow the main window.
increase_ratio()	Increase spiral ratio.
items(name)	Build a list of contained items for the given item class.
next()	
previous()	
reset()	Reset ratios to values set in config.
set_master_ratio(ratio)	Set the ratio for the main window.
set_ratio(ratio)	Set the ratio for all windows.
shrink_main()	Shrink the main window.
shuffle_down()	
shuffle_up()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_ratio()

Decrease spiral ratio.

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

grow_main()

Grow the main window.

increase_ratio()

Increase spiral ratio.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next() → None

previous() → None

reset()

Reset ratios to values set in config.

set_master_ratio(*ratio: float*)

Set the ratio for the main window.

set_ratio(*ratio: float*)

Set the ratio for all windows.

shrink_main()

Shrink the main window.

shuffle_down()

shuffle_up()

15.2.13 Stack

class libqtile.layout.stack.**Stack**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

add()	Add another stack to the layout
client_to_next()	Send the current client to the next stack
client_to_previous()	Send the current client to the previous stack
client_to_stack(n)	Send the current client to stack n, where n is an integer offset.
commands()	Returns a list of possible commands for this object
delete()	Delete the current stack from the layout
doc(name)	Returns the documentation for a specified command name
down()	Switch to the next window in this stack
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
next()	Focus next stack
previous()	Focus previous stack
rotate()	Rotate order of the stacks
shuffle_down()	Shuffle the order of this stack down
shuffle_up()	Shuffle the order of this stack up
toggle_split()	Toggle vertical split on the current stack
up()	Switch to the previous window in this stack

Command documentation

add()

Add another stack to the layout

client_to_next()

Send the current client to the next stack

client_to_previous()

Send the current client to the previous stack

client_to_stack(*n*)

Send the current client to stack n, where n is an integer offset. If is too large or less than 0, it is wrapped modulo the number of stacks.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

delete()

Delete the current stack from the layout

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

down()

Switch to the next window in this stack

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next() → None

Focus next stack

previous() → None

Focus previous stack

rotate()

Rotate order of the stacks

shuffle_down()

Shuffle the order of this stack down

shuffle_up()

Shuffle the order of this stack up

toggle_split()

Toggle vertical split on the current stack

up()

Switch to the previous window in this stack

15.2.14 Tile

class libqtile.layout.tile.**Tile**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
decrease_nmaster()	
decrease_ratio()	
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
increase_nmaster()	
increase_ratio()	
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
next()	
previous()	
reset()	
shuffle_down()	
shuffle_up()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_nmaster()

decrease_ratio()

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

increase_nmaster()

increase_ratio()

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next() → None

previous() → None

reset()

shuffle_down()

shuffle_up()

15.2.15 TreeTab

class libqtile.layout.tree.**TreeTab**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

add_section(name)	Add named section to tree
collapse_branch()	
commands()	Returns a list of possible commands for this object
decrease_ratio()	
del_section(name)	Remove named section from tree
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
expand_branch()	
function(function, *args, **kwargs)	Call a function with current object as argument
increase_ratio()	
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
move_down()	
move_left()	
move_right()	
move_up()	
next()	Switch down in the window list
previous()	Switch up in the window list
section_down()	
section_up()	
sort_windows(sorter[, create_sections])	Sorts window to sections using sorter function

Command documentation

add_section(*name*)

Add named section to tree

collapse_branch()

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_ratio()

del_section(*name*)

Remove named section from tree

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

expand_branch()

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

increase_ratio()

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

move_down()

move_left()

move_right()

move_up()

next() → None

Switch down in the window list

previous() → None

Switch up in the window list

section_down()

section_up()

sort_windows(*sorter*, *create_sections=True*)

Sorts window to sections using sorter function

Parameters

sorter: function with single arg returning string
returns name of the section where window should be

create_sections:
if this parameter is True (default), if sorter returns unknown section name it will be created dynamically

15.2.16 VerticalTile

class libqtile.layout.verticaltile.**VerticalTile**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
grow()	
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
maximize()	
next()	
normalize()	
previous()	
shrink()	
shuffle_down()	
shuffle_up()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None
Call a function with current object as argument

grow()

info() → dict[str, Any]
Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]
Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

maximize()

next() → None

normalize()

previous() → None

shrink()

shuffle_down()

shuffle_up()

15.2.17 Zoomy

class libqtile.layout.zoomy.**Zoomy**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.layout.<command>()
qtile cmd-obj -o layout -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of layout information
items(name)	Build a list of contained items for the given item class.
next()	
previous()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Returns a dictionary of layout information

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

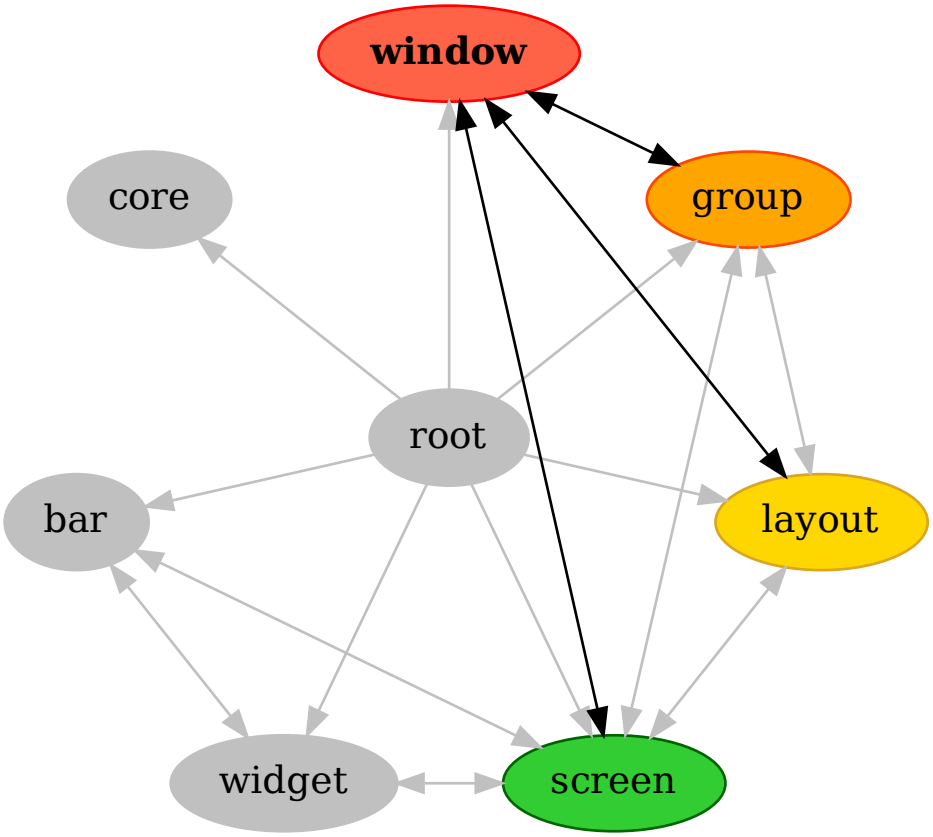
next() → None

previous() → None

15.3 Window objects

The size and position of windows is determined by the current layout. Nevertheless, windows can still change their appearance in multiple ways (toggling floating state, fullscreen, opacity).

Windows can access objects relevant to the display of the window (i.e. the screen, group and layout).



class libqtile.backend.base.window.**Window**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.window.<command>()
qtile cmd-obj -o window -f <command>

The following commands are available for this object:

bring_to_front()	Bring the window to the front.
center()	Centers a floating window on the screen.
commands()	Returns a list of possible commands for this object
disable_floating()	Tile the window.

continues on next page

Table 2 – continued from previous page

disable_fullscreen()	Un-fullscreen the window
doc(name)	Returns the documentation for a specified command name
down_opacity()	Decrease the window's opacity by 10%.
enable_floating()	Float the window.
enable_fullscreen()	Fullscreen the window
eval(code)	Evaluates code in the same context as this function
focus([warp])	Focus this window and optional warp the pointer to it.
function(function, *args, **kwargs)	Call a function with current object as argument
get_position()	Get the (x, y) of the window
get_size()	Get the (width, height) of the window
info()	Return information on this window.
is_visible()	Is this window visible (i.e.
items(name)	Build a list of contained items for the given item class.
keep_above([enable])	Keep this window above all others
keep_below([enable])	Keep this window below all others
kill()	Kill the window
move_down([force])	Move this window below the previous window along the z axis.
move_floating(dx, dy)	Move window by dx and dy
move_to_bottom()	Move this window below all windows in the current layer e.g.
move_to_top()	Move this window above all windows in the current layer e.g.
move_up([force])	Move this window above the next window along the z axis.
place(x, y, width, height, borderwidth, ...)	Place the window in the given position.
resize_floating(dw, dh)	Add dw and dh to size of window
set_opacity(opacity)	Set the window's opacity
set_position(x, y)	Move floating window to x and y; swap tiling window with the window under the pointer.
set_position_floating(x, y)	Move window to x and y
set_size_floating(w, h)	Set window dimensions to w and h
static([screen, x, y, width, height])	Makes this window a static window, attached to a Screen.
toggle_floating()	Toggle the floating state of the window.
toggle_fullscreen()	Toggle the fullscreen state of the window.
toggle_maximize()	Toggle the fullscreen state of the window.
toggle_minimize()	Toggle the minimized state of the window.
togroup([group_name, groupName, ...])	Move window to a specified group
toscreen([index])	Move window to a specified screen.
up_opacity()	Increase the window's opacity by 10%.

Command documentation

abstract bring_to_front() → None

Bring the window to the front.

In X11, *bring_to_front* ignores all other layering rules and brings the window to the very front. When that window loses focus, it will be stacked again according the appropriate rules.

center() → None

Centers a floating window on the screen.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

abstract disable_floating() → None

Tile the window.

abstract disable_fullscreen() → None

Un-fullscreen the window

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

down_opacity() → None

Decrease the window's opacity by 10%.

abstract enable_floating() → None

Float the window.

abstract enable_fullscreen() → None

Fullscreen the window

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

abstract focus(*warp: bool = True*) → None

Focus this window and optional warp the pointer to it.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

abstract get_position() → tuple[int, int]

Get the (x, y) of the window

abstract get_size() → tuple[int, int]

Get the (width, height) of the window

abstract info() → dict[str, Any]

Return information on this window.

Mimimum required keys are: - name - x - y - width - height - group - id - wm_class

is_visible() → bool

Is this window visible (i.e. not hidden)?

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

keep_above(*enable: bool | None = None*)

Keep this window above all others

keep_below(*enable: bool | None = None*)

Keep this window below all others

abstract kill() → None

Kill the window

move_down(*force: bool = False*) → None

Move this window below the previous window along the z axis.

Will not lower a "normal" window (i.e. one that is not "kept_above/below") below a window that is marked as "kept_below".

Will not lower a window where "keep_above" is True unless force is set to True.

abstract move_floating(*dx: int*, *dy: int*) → None

Move window by dx and dy

move_to_bottom() → None

Move this window below all windows in the current layer e.g. if you have 3 windows all with "keep_above" set, calling this method will move the window to the bottom of those three windows.

Calling this on a "normal" window will not raise it below a "kept_below" window.

move_to_top() → None

Move this window above all windows in the current layer e.g. if you have 3 windows all with "keep_above" set, calling this method will move the window to the top of those three windows.

Calling this on a "normal" window will not raise it above a "kept_above" window.

move_up(*force: bool = False*) → None

Move this window above the next window along the z axis.

Will not raise a "normal" window (i.e. one that is not "kept_above/below") above a window that is marked as "kept_above".

Will not raise a window where "keep_below" is True unless force is set to True.

abstract place(*x*, *y*, *width*, *height*, *borderwidth*, *bordercolor*, *above=False*, *margin=None*, *respect_hints=False*)

Place the window in the given position.

abstract resize_floating(*dw: int*, *dh: int*) → None

Add dw and dh to size of window

set_opacity(*opacity: float*) → None

Set the window's opacity

abstract set_position(*x: int*, *y: int*) → None

Move floating window to x and y; swap tiling window with the window under the pointer.

abstract set_position_floating(*x: int*, *y: int*) → None

Move window to x and y

abstract set_size_floating(*w: int*, *h: int*) → None

Set window dimensions to w and h

abstract static(*screen: int | None = None*, *x: int | None = None*, *y: int | None = None*, *width: int | None = None*, *height: int | None = None*) → None

Makes this window a static window, attached to a Screen.

Values left unspecified are taken from the existing window state.

abstract toggle_floating() → None

Toggle the floating state of the window.

abstract toggle_fullscreen() → None

Toggle the fullscreen state of the window.

abstract toggle_maximize() → None

Toggle the fullscreen state of the window.

abstract toggle_minimize() → None

Toggle the minimized state of the window.

abstract togroup(*group_name: str | None = None*, *groupName: str | None = None*, *switch_group: bool = False*, *toggle: bool = False*) → None

Move window to a specified group

Also switch to that group if *switch_group* is True.

If *toggle* is True and and the specified group is already on the screen, use the last used group as target instead.

groupName is deprecated and will be dropped soon. Please use *group_name* instead.

Examples

Move window to current group:

```
togroup()
```

Move window to group "a":

```
togroup("a")
```

Move window to group "a", and switch to group "a":

```
togroup("a", switch_group=True)
```

toscreen(*index: int | None = None*) → None

Move window to a specified screen.

If index is not specified, we assume the current screen

Examples

Move window to current screen:

```
toscreen()
```

Move window to screen 0:

```
toscreen(0)
```

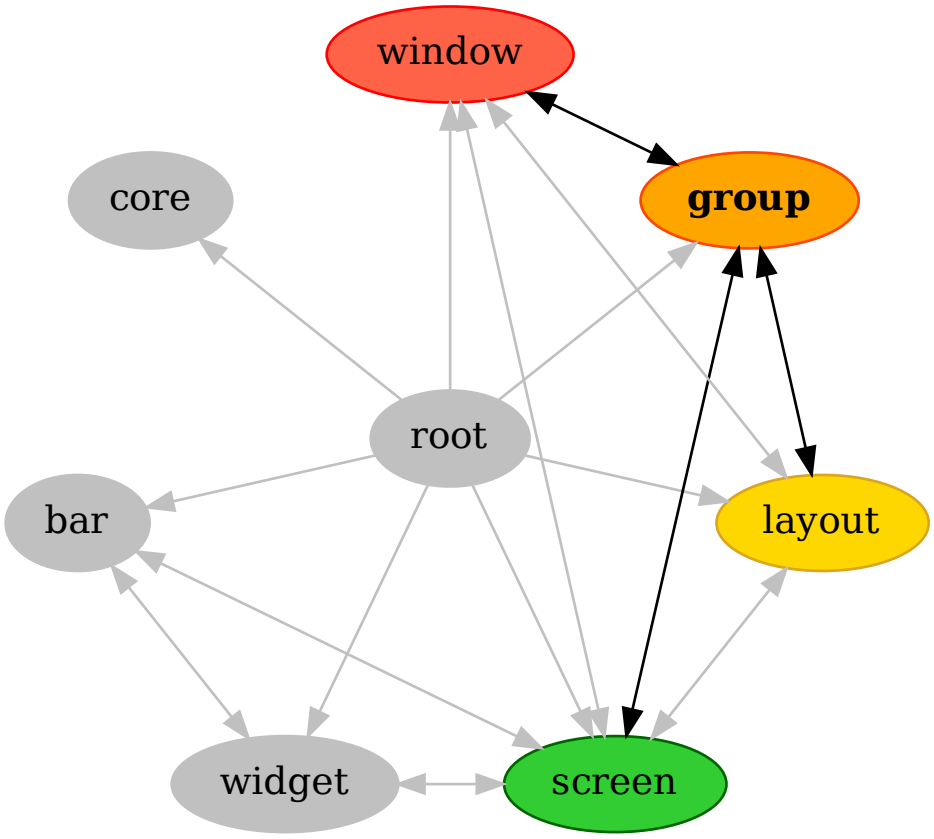
up_opacity() → None

Increase the window's opacity by 10%.

15.4 Group objects

Groups are Qtile's workspaces. Groups are not responsible for the positioning of windows (that is handled by the *layouts*) so the available commands are somewhat more limited in scope.

Groups have access to the layouts in that group, the windows in the group and the screen displaying the group.



class libqtile.group._Group

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.group.<command>()
qtile cmd-obj -o group -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
focus_back()	Focus the window that had focus before the current one got it.
focus_by_index(index)	Change to the window at the specified index in the current group.
focus_by_name(name)	Focus the first window with the given name.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of info for this group
info_by_name(name)	Get the info for the first window with the given name without giving it focus.
items(name)	Build a list of contained items for the given item class.
next_window()	Focus the next window in group.
prev_window()	Focus the previous window in group.
set_label(label)	Set the display name of current group to be used in GroupBox widget.
setlayout(layout)	
swap_window_order(new_location)	Change the order of the current window within the current group.
switch_groups(name)	Switch position of current group with name
toscreen([screen, toggle])	Pull a group to a specified screen.
unminimize_all()	Unminimise all windows in this group

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

focus_back()

Focus the window that had focus before the current one got it.

Repeated calls to this function would basically continuously switch between the last two focused windows. Do nothing if less than 2 windows ever received focus.

focus_by_index(*index: int*) → None

Change to the window at the specified index in the current group.

focus_by_name(*name*)

Focus the first window with the given name. Do nothing if the name is not found.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Returns a dictionary of info for this group

info_by_name(*name*)

Get the info for the first window with the given name without giving it focus. Do nothing if the name is not found.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next_window()

Focus the next window in group.

Method cycles _all_ windows in group regardless if tiled in current layout or floating. Cycling of tiled and floating windows is not mixed. The cycling order depends on the current Layout.

prev_window()

Focus the previous window in group.

Method cycles _all_ windows in group regardless if tiled in current layout or floating. Cycling of tiled and floating windows is not mixed. The cycling order depends on the current Layout.

set_label(*label*)

Set the display name of current group to be used in GroupBox widget. If label is None, the name of the group is used as display name. If label is the empty string, the group is invisible in GroupBox.

setlayout(*layout*)

swap_window_order(*new_location: int*) → None

Change the order of the current window within the current group.

switch_groups(*name*)

Switch position of current group with name

toscreen(*screen=None*, *toggle=False*)

Pull a group to a specified screen.

Parameters

screen
Screen offset. If not specified, we assume the current screen.

toggle
If this group is already on the screen, then the group is toggled with last used

Examples

Pull group to the current screen:

```
toscreen()
```

Pull group to screen 0:

```
toscreen(0)
```

unminimize_all()

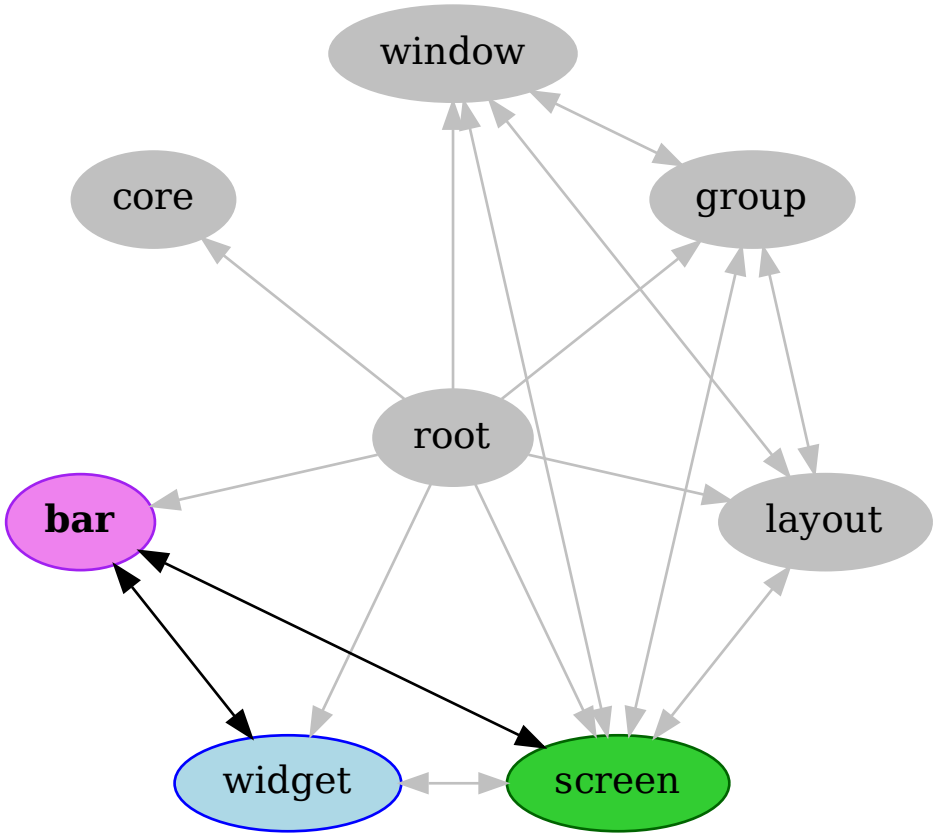
Unminimise all windows in this group

15.5 Bar objects

The bar is primarily used to display widgets on the screen. As a result, the bar does not need many of its own commands.

To select a bar on the command graph, you must use a selector (as there is no default bar). The selector is the position of the bar on the screen i.e. "top", "bottom", "left" or "right".

The bar can access the screen it's on and the widgets it contains via the command graph.



class libqtile.bar.**Bar**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.bar["position"].<command>()
qtile cmd-obj -o bar position -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
fake_button_press(screen, position, x, y[, ...])	Fake a mouse-button-press on the bar.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

fake_button_press(*screen: int*, *position: str*, *x: int*, *y: int*, *button: int = 1*) → None

Fake a mouse-button-press on the bar. Co-ordinates are relative to the top-left corner of the bar.

Parameters

widgets
A list of widget objects.

size
The "thickness" of the bar, i.e. the height of a horizontal bar, or the width of a vertical bar.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

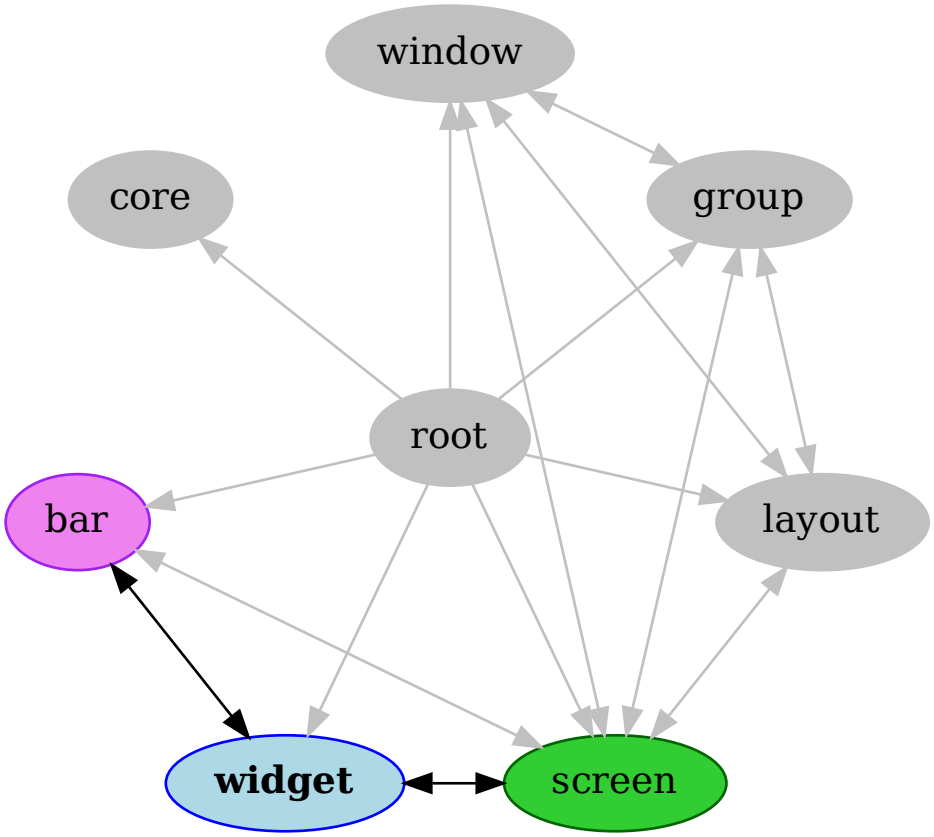
root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6 Widget objects

Widgets are small scripts that are used to provide content or add functionality to the bar. Some widgets will expose commands in order for functionality to be triggered indirectly (e.g. via a keypress).

Widgets can access the parent bar and screen via the command graph.



15.6.1 AGroupBox

class libqtile.widget.groupbox.**AGroupBox**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["agroupbox"].<command>()
qtile cmd-obj -o widget agroupbox -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.2 Backlight

class libqtile.widget.backlight.**Backlight**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["backlight"].<command>()
qtile cmd-obj -o widget backlight -f <command>
```

The following commands are available for this object:

change_backlight(direction[, step])	
commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

change_backlight(*direction*, *step=None*)

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.3 Battery

class libqtile.widget.battery.**Battery**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["battery"].<command>()
qtile cmd-obj -o widget battery -f <command>

The following commands are available for this object:

charge_dynamically()	
charge_to_full()	
commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

charge_dynamically()

charge_to_full()

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.4 BatteryIcon

class libqtile.widget.battery.**BatteryIcon**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["batteryicon"].<command>()
qtile cmd-obj -o widget batteryicon -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.5 Bluetooth

class libqtile.widget.bluetooth.**Bluetooth**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["bluetooth"].<command>()
qtile cmd-obj -o widget bluetooth -f <command>

The following commands are available for this object:

click()	Perform default action on visible item.
commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
scroll_down()	Scroll down to next item.
scroll_up()	Scroll up to next item.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

click()

Perform default action on visible item.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

scroll_down()

Scroll down to next item.

scroll_up()

Scroll up to next item.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.6 CPU

class libqtile.widget.cpu.**CPU**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["cpu"].<command>()
qtile cmd-obj -o widget cpu -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.7 CPUGraph

class libqtile.widget.graph.**CPUGraph**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["cpugraph"].<command>()
qtile cmd-obj -o widget cpugraph -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.8 Canto

class libqtile.widget.canto.**Canto**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["canto"].<command>()
qtile cmd-obj -o widget canto -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.9 CapsNumLockIndicator

class libqtile.widget.caps_num_lock_indicator.**CapsNumLockIndicator**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["capsnumlockindicator"].<command>()
qtile cmd-obj -o widget capsnumlockindicator -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.10 CheckUpdates

class libqtile.widget.check_updates.**CheckUpdates**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["checkupdates"].<command>()
qtile cmd-obj -o widget checkupdates -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.11 Chord

class libqtile.widget.chord.**Chord**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["chord"].<command>()
qtile cmd-obj -o widget chord -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.12 Clipboard

class libqtile.widget.clipboard.**Clipboard**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["clipboard"].<command>()
qtile cmd-obj -o widget clipboard -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.13 Clock

class libqtile.widget.clock.**Clock**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["clock"].<command>()
qtile cmd-obj -o widget clock -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
update_timezone([timezone])	Force the clock to update timezone information.
use_system_timezone()	Force clock to use system timezone.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

update_timezone(*timezone: str | tzinfo | None = None*)

Force the clock to update timezone information.

If the method is called with no arguments then the widget will reload the timzeone set on the computer (e.g. via `timedatectl set-timezone ..`). This will have no effect if you have previously set a `timezone` value.

Alternatively, you can pass a timezone string (e.g. `"Europe/Lisbon"`) to change the specified timezone. Setting this to an empty string will cause the clock to rely on the system timezone.

use_system_timezone()

Force clock to use system timezone.

15.6.14 Cmus

class libqtile.widget.cmus.**Cmus**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["cmus"].<command>()
qtile cmd-obj -o widget cmus -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.15 Countdown

class libqtile.widget.countdown.**Countdown**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["countdown"].<command>()
qtile cmd-obj -o widget countdown -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.16 CryptoTicker

class libqtile.widget.crypto_ticker.**CryptoTicker**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["cryptoticker"].<command>()
qtile cmd-obj -o widget cryptoticker -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.17 CurrentLayout

class libqtile.widget.currentlayout.**CurrentLayout**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["currentlayout"].<command>()
qtile cmd-obj -o widget currentlayout -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.18 CurrentLayoutIcon

class libqtile.widget.currentlayout.**CurrentLayoutIcon**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["currentlayouticon"].<command>()
qtile cmd-obj -o widget currentlayouticon -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.19 CurrentScreen

class libqtile.widget.currentscreen.**CurrentScreen**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["currentscreen"].<command>()
qtile cmd-obj -o widget currentscreen -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.20 DF

class libqtile.widget.df.**DF**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["df"].<command>()
qtile cmd-obj -o widget df -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.21 DoNotDisturb

class libqtile.widget.do_not_disturb.**DoNotDisturb**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["donotdisturb"].<command>()
qtile cmd-obj -o widget donotdisturb -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.22 GenPollCommand

class libqtile.widget.generic_poll_text.**GenPollCommand**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["genpollcommand"].<command>()
qtile cmd-obj -o widget genpollcommand -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.23 GenPollText

class libqtile.widget.generic_poll_text.**GenPollText**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["genpolltext"].<command>()
qtile cmd-obj -o widget genpolltext -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.24 GenPollUrl

class libqtile.widget.generic_poll_text.**GenPollUrl**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["genpollurl"].<command>()
qtile cmd-obj -o widget genpollurl -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.25 GmailChecker

class libqtile.widget.gmail_checker.**GmailChecker**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["gmailchecker"].<command>()
qtile cmd-obj -o widget gmailchecker -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.26 GroupBox

class libqtile.widget.groupbox.**GroupBox**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["groupbox"].<command>()
qtile cmd-obj -o widget groupbox -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.27 HDDBusyGraph

class libqtile.widget.graph.**HDDBusyGraph**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["hddbusygraph"].<command>()
qtile cmd-obj -o widget hddbusygraph -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.28 HDDGraph

class libqtile.widget.graph.**HDDGraph**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["hddgraph"].<command>()
qtile cmd-obj -o widget hddgraph -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.29 IdleRPG

class libqtile.widget.idlerpg.**IdleRPG**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["idlerpg"].<command>()
qtile cmd-obj -o widget idlerpg -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.30 Image

class libqtile.widget.image.**Image**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["image"].<command>()
qtile cmd-obj -o widget image -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
update(filename)	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

update(*filename*)

15.6.31 ImapWidget

class libqtile.widget.imapwidget.**ImapWidget**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["imapwidget"].<command>()
qtile cmd-obj -o widget imapwidget -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.32 KeyboardKbdd

class libqtile.widget.keyboardkbdd.**KeyboardKbdd**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["keyboardkbdd"].<command>()
qtile cmd-obj -o widget keyboardkbdd -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.33 KeyboardLayout

class libqtile.widget.keyboardlayout.**KeyboardLayout**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["keyboardlayout"].<command>()
qtile cmd-obj -o widget keyboardlayout -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
next_keyboard()	set the next layout in the list of configured keyboard layouts as new current layout in use
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next_keyboard()

set the next layout in the list of configured keyboard layouts as new current layout in use

If the current keyboard layout is not in the list, it will set as new layout the first one in the list.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.34 KhalCalendar

class libqtile.widget.khal_calendar.**KhalCalendar**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["khalcalendar"].<command>()
qtile cmd-obj -o widget khalcalendar -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.35 LaunchBar

class libqtile.widget.launchbar.**LaunchBar**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["launchbar"].<command>()
qtile cmd-obj -o widget launchbar -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.36 Load

class libqtile.widget.load.**Load**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["load"].<command>()
qtile cmd-obj -o widget load -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
next_load()	
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next_load()

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.37 Maildir

class libqtile.widget.maildir.**Maildir**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["maildir"].<command>()
qtile cmd-obj -o widget maildir -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.38 Memory

class libqtile.widget.memory.**Memory**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["memory"].<command>()
qtile cmd-obj -o widget memory -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.39 MemoryGraph

class libqtile.widget.graph.**MemoryGraph**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["memorygraph"].<command>()
qtile cmd-obj -o widget memorygraph -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.40 Mirror

class libqtile.widget.base.**Mirror**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["mirror"].<command>()
qtile cmd-obj -o widget mirror -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.41 Moc

class libqtile.widget.moc.**Moc**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["moc"].<command>()
qtile cmd-obj -o widget moc -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.42 Mpd2

class libqtile.widget.mpd2widget.**Mpd2**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["mpd2"].<command>()
qtile cmd-obj -o widget mpd2 -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.43 Mpris2

class libqtile.widget.mpris2widget.**Mpris2**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["mpris2"].<command>()
qtile cmd-obj -o widget mpris2 -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	What's the current state of the widget?
items(name)	Build a list of contained items for the given item class.
next()	Play the next track.
play_pause()	Toggle the playback status.
previous()	Play the previous track.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
stop()	Stop playback.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

What's the current state of the widget?

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next() → None

Play the next track.

play_pause() → None

Toggle the playback status.

previous() → None

Play the previous track.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

stop() → None

Stop playback.

15.6.44 Net

class libqtile.widget.net.**Net**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["net"].<command>()
qtile cmd-obj -o widget net -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.45 NetGraph

class libqtile.widget.graph.**NetGraph**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["netgraph"].<command>()
qtile cmd-obj -o widget netgraph -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.46 Notify

class libqtile.widget.notify.**Notify**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["notify"].<command>()
qtile cmd-obj -o widget notify -f <command>
```

The following commands are available for this object:

clear([reason])	Clear the notification
commands()	Returns a list of possible commands for this object
display()	
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
invoke()	Invoke the notification's default action
items(name)	Build a list of contained items for the given item class.
next()	
prev()	Show previous notification.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
toggle()	Toggle showing/clearing the notification

Command documentation

clear(*reason=2*)

Clear the notification

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

display()

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

invoke()

Invoke the notification's default action

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next()

prev()

Show previous notification.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

toggle()

Toggle showing/clearing the notification

15.6.47 NvidiaSensors

class libqtile.widget.nvidia_sensors.**NvidiaSensors**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["nvidiasensors"].<command>()
qtile cmd-obj -o widget nvidiasensors -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.48 OpenWeather

class libqtile.widget.open_weather.**OpenWeather**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["openweather"].<command>()
qtile cmd-obj -o widget openweather -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.49 Pomodoro

class libqtile.widget.pomodoro.**Pomodoro**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["pomodoro"].<command>()
qtile cmd-obj -o widget pomodoro -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
toggle_active()	
toggle_break()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

toggle_active()

toggle_break()

15.6.50 Prompt

class libqtile.widget.prompt.**Prompt**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["prompt"].<command>()
qtile cmd-obj -o widget prompt -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
exec_general(prompt, object_name, cmd_name)	Execute a cmd of any object.
fake_keypress(key)	
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of info for this object
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

exec_general(*prompt*, *object_name*, *cmd_name*, *selector=None*, *completer=None*)

Execute a cmd of any object. For example layout, group, window, widget , etc with a string that is obtained from start_input.

Parameters

prompt
Text displayed at the prompt.

object_name
Name of a object in Qtile. This string has to be 'layout', 'widget', 'bar', 'window' or 'screen'.

cmd_name
Execution command of selected object using object_name and selector.

selector
This value select a specific object within a object list that is obtained by object_name. If this value is None, current object is selected. e.g. current layout, current window and current screen.

completer:
Completer to use.

config example:

Key([alt, 'shift'], 'a',

lazy.widget['prompt'].exec_general(
'section(add)', 'layout', 'add_section'))

`fake_keypress`(*key: str*) → None

`function`(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

`info`()

Returns a dictionary of info for this object

`items`(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

`set_font`(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.51 PulseVolume

`class` libqtile.widget.pulse_volume.`PulseVolume`

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["pulsevolume"].<command>()
qtile cmd-obj -o widget pulsevolume -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
decrease_vol([value])	Decrease volume.
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
increase_vol([value])	Increase volume.
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
mute()	Mute the sound device.
run_app()	
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_vol(*value=None*)

Decrease volume.

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

increase_vol(*value=None*)

Increase volume.

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

mute()

Mute the sound device.

run_app()

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.52 QuickExit

class libqtile.widget.quick_exit.**QuickExit**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["quickexit"].<command>()
qtile cmd-obj -o widget quickexit -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
trigger()	

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

trigger()

15.6.53 ScreenSplit

class libqtile.widget.screensplit.**ScreenSplit**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["screensplit"].<command>()
qtile cmd-obj -o widget screensplit -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
get()	Retrieve the text in a TextBox widget
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
update(text)	Update the widget text.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

get()

Retrieve the text in a TextBox widget

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

update(*text*)

Update the widget text.

15.6.54 Sep

class libqtile.widget.sep.**Sep**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["sep"].<command>()
qtile cmd-obj -o widget sep -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.55 She

class libqtile.widget.she.**She**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["she"].<command>()
qtile cmd-obj -o widget she -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.56 Spacer

class libqtile.widget.spacer.**Spacer**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["spacer"].<command>()
qtile cmd-obj -o widget spacer -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.57 StatusNotifier

class libqtile.widget.statusnotifier.**StatusNotifier**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["statusnotifier"].<command>()
qtile cmd-obj -o widget statusnotifier -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.58 StockTicker

class libqtile.widget.stock_ticker.**StockTicker**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["stockticker"].<command>()
qtile cmd-obj -o widget stockticker -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.59 SwapGraph

class libqtile.widget.graph.**SwapGraph**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["swapgraph"].<command>()
qtile cmd-obj -o widget swapgraph -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.60 Systray

class libqtile.widget.systray.**Systray**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["systray"].<command>()
qtile cmd-obj -o widget systray -f <command>

The following commands are available for this object:

_window_mask	
commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
focus([warp])	Focuses the window.
function(function, *args, **kwargs)	Call a function with current object as argument
get_hints()	Returns the X11 hints (WM_HINTS and WM_SIZE_HINTS) for this window.
inspect()	Tells you more than you ever wanted to know about a window
items(name)	Build a list of contained items for the given item class.
keep_above([enable])	
keep_below([enable])	
kill()	
move_down([force])	
move_to_bottom([force])	
move_to_top([force])	
move_up([force])	
place(x, y, width, height, borderwidth, ...)	Places the window at the specified location with the given size.

Command documentation

_window_mask(***kwargs*)

int([x]) -> integer int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments are given. If x is a number, return x.__int__(). For floating point numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string, bytes, or bytearray instance representing an integer literal in the given base. The literal can be preceded by '+' or '-' and be surrounded by whitespace. The base defaults to 10. Valid bases are 0 and 2-36. Base 0 means to interpret the base from the string as an integer literal. >>> int('0b100', base=0) 4

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

focus(*warp: bool = True*) → None

Focuses the window.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

get_hints()

Returns the X11 hints (WM_HINTS and WM_SIZE_HINTS) for this window.

inspect()

Tells you more than you ever wanted to know about a window

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

keep_above(*enable: bool | None = None*)

keep_below(*enable: bool | None = None*)

kill()

move_down(*force=False*)

move_to_bottom(*force=False*)

move_to_top(*force=False*)

move_up(*force=False*)

place(*x*, *y*, *width*, *height*, *borderwidth*, *bordercolor*, *above=False*, *margin=None*, *respect_hints=False*)

Places the window at the specified location with the given size.

Parameters

x: int
y: int
width: int
height: int
borderwidth: int
bordercolor: string
above: bool, optional
margin: int or list, optional
space around window as int or list of ints [N E S W]

above: bool, optional
If True, the geometry will be adjusted to respect hints provided by the client.

15.6.61 TaskList

class libqtile.widget.tasklist.**TaskList**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["tasklist"].<command>()
qtile cmd-obj -o widget tasklist -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.6.62 TextBox

class libqtile.widget.textbox.**TextBox**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["textbox"].<command>()
qtile cmd-obj -o widget textbox -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
get()	Retrieve the text in a TextBox widget
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
update(text)	Update the widget text.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

get()

Retrieve the text in a TextBox widget

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

update(*text*)

Update the widget text.

15.6.63 ThermalSensor

class libqtile.widget.sensors.**ThermalSensor**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["thermalsensor"].<command>()
qtile cmd-obj -o widget thermalsensor -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.64 ThermalZone

class libqtile.widget.thermal_zone.**ThermalZone**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["thermalzone"].<command>()
qtile cmd-obj -o widget thermalzone -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.65 Volume

class libqtile.widget.volume.**Volume**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["volume"].<command>()
qtile cmd-obj -o widget volume -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
decrease_vol()	
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
increase_vol()	
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
mute()	
run_app()	
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_vol()

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

increase_vol()

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

mute()

run_app()

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.66 Wallpaper

class libqtile.widget.wallpaper.**Wallpaper**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["wallpaper"].<command>()
qtile cmd-obj -o widget wallpaper -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.67 WidgetBox

class libqtile.widget.widgetbox.**WidgetBox**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["widgetbox"].<command>()
qtile cmd-obj -o widget widgetbox -f <command>
```

The following commands are available for this object:

close()	Close the widgetbox.
commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
open()	Open the widgetbox.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.
toggle()	Toggle box state

Command documentation

close()

Close the widgetbox.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

open()

Open the widgetbox.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

toggle()

Toggle box state

15.6.68 WindowCount

class libqtile.widget.window_count.**WindowCount**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["windowcount"].<command>()
qtile cmd-obj -o widget windowcount -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
get()	Retrieve the current text.
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None
Call a function with current object as argument

get()
Retrieve the current text.

info()
Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]
Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)
Change the font used by this widget. If font is None, the current font is used.

15.6.69 WindowName

class libqtile.widget.windowname.**WindowName**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["windowname"].<command>()
qtile cmd-obj -o widget windowname -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]
Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str
Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.70 WindowTabs

class libqtile.widget.windowtabs.**WindowTabs**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["windowtabs"].<command>()
qtile cmd-obj -o widget windowtabs -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.71 Wlan

class libqtile.widget.wlan.**Wlan**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.widget["wlan"].<command>()
qtile cmd-obj -o widget wlan -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.6.72 Wttr

class libqtile.widget.wttr.**Wttr**

API commands

To access commands on this object via the command graph, use one of the following options:

```
lazy.widget["wttr"].<command>()
qtile cmd-obj -o widget wttr -f <command>
```

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
force_update()	Immediately poll the widget.
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Info for this object.
items(name)	Build a list of contained items for the given item class.
set_font([font, fontsize, fontshadow])	Change the font used by this widget.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

force_update()

Immediately poll the widget. Existing timers are unaffected.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

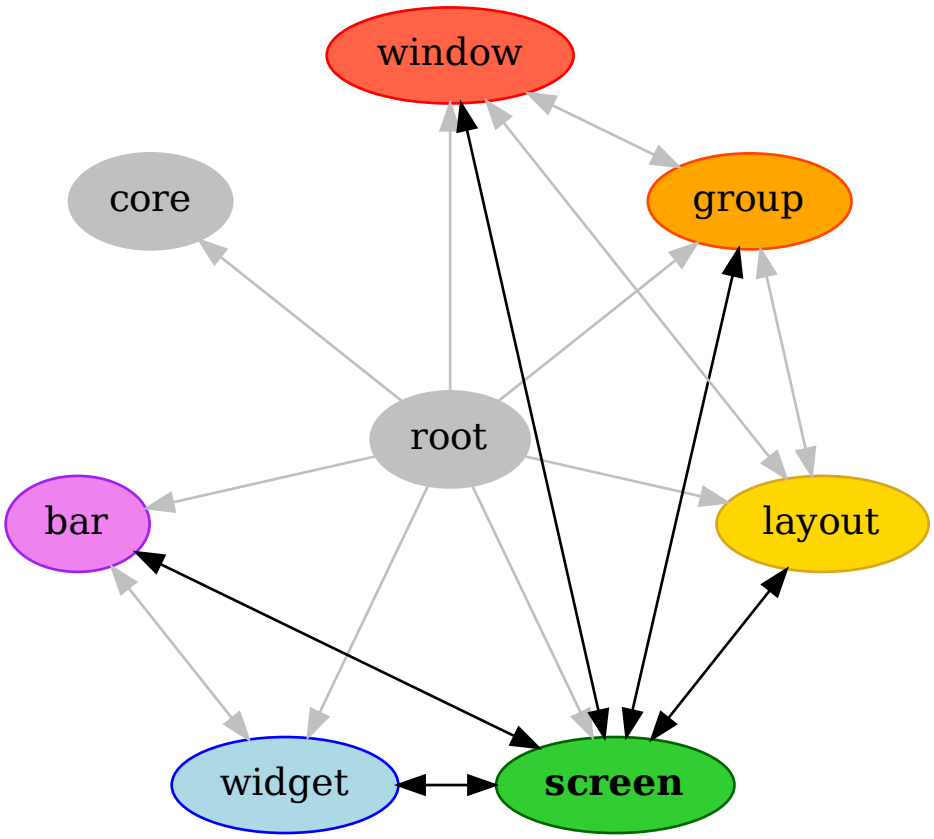
set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

15.7 Screen objects

Screens are the display area that holds bars and an active group. Screen commands include changing the current group and changing the wallpaper.

Screens can access objects displayed on that screen e.g. bar, widgets, groups, layouts and windows.



class libqtile.config.**Screen**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.screen.<command>()
qtile cmd-obj -o screen -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Returns a dictionary of info for this screen.
items(name)	Build a list of contained items for the given item class.
next_group([skip_empty, skip_managed])	Switch to the next group
prev_group([skip_empty, skip_managed, warp])	Switch to the previous group
resize([x, y, w, h])	
set_wallpaper(path[, mode])	Set the wallpaper to the given file.
toggle_group([group_name, warp])	Switch to the selected group or to the previously active one

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, int]

Returns a dictionary of info for this screen.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next_group(*skip_empty: bool = False*, *skip_managed: bool = False*) → None

Switch to the next group

prev_group(*skip_empty: bool = False*, *skip_managed: bool = False*, *warp: bool = True*) → None

Switch to the previous group

resize(*x: int | None = None*, *y: int | None = None*, *w: int | None = None*, *h: int | None = None*) → None

set_wallpaper(*path: str*, *mode: str | None = None*) → None

Set the wallpaper to the given file.

toggle_group(*group_name: str | None = None*, *warp: bool = True*) → None

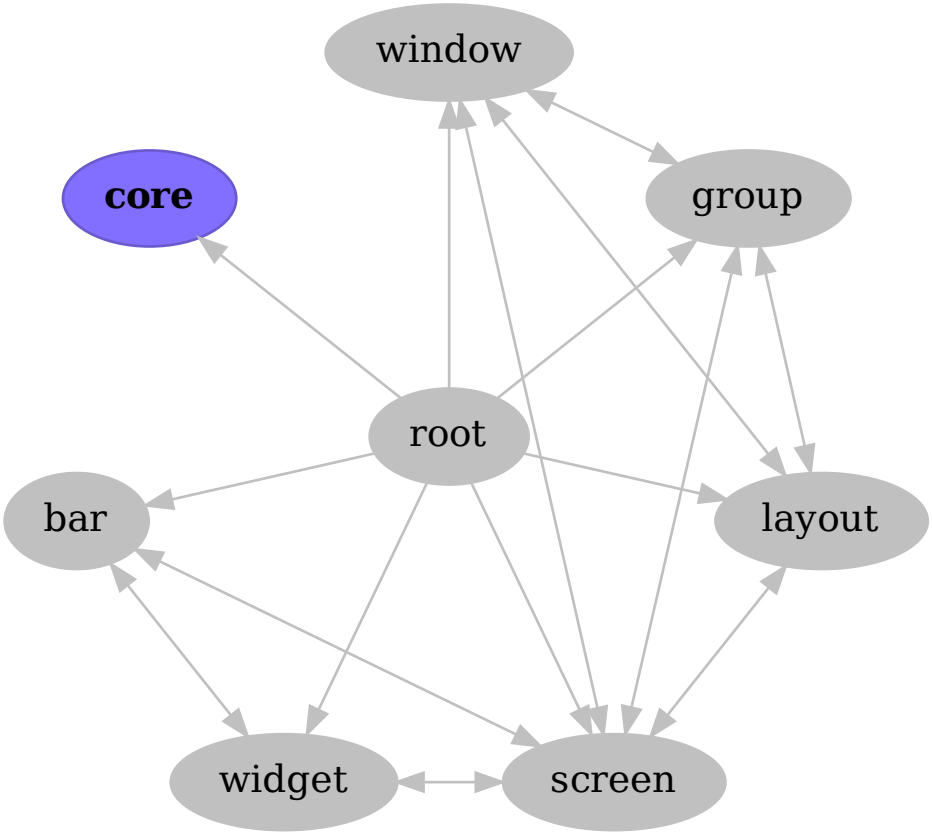
Switch to the selected group or to the previously active one

15.8 Backend core objects

The backend core is the link between the Qtile objects (windows, layouts, groups etc.) and the specific backend (X11 or Wayland). This core should be largely invisible to users and, as a result, these objects do not expose many commands.

Nevertheless, both backends do contain important commands, notably `set_keymap` on X11 and `change_vt` used to change to a different TTY on Wayland.

The backend core has no access to other nodes on the command graph.



15.8.1 X11 backend

class libqtile.backend.x11.core.**Core**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.core.<command>()
qtile cmd-obj -o core -f <command>

The following commands are available for this object:

commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
info()	Get basic information about the running backend.
items(name)	Build a list of contained items for the given item class.

Command documentation

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info() → dict[str, Any]

Get basic information about the running backend.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

15.8.2 Wayland backend

class libqtile.backend.wayland.core.**Core**

API commands

To access commands on this object via the command graph, use one of the following options:

lazy.core.<command>()
qtile cmd-obj -o core -f <command>

The following commands are available for this object:

change_vt(vt)	Change virtual terminal to that specified
commands()	Returns a list of possible commands for this object
doc(name)	Returns the documentation for a specified command name
eval(code)	Evaluates code in the same context as this function
function(function, *args, **kwargs)	Call a function with current object as argument
get_inputs()	Get information on all input devices.
hide_cursor()	Hide the cursor.
info()	Get basic information about the running backend.
items(name)	Build a list of contained items for the given item class.
query_tree()	Get IDs of all mapped windows in ascending Z order.
set_keymap([layout, options, variant])	Set the keymap for the current keyboard.
unhide_cursor()	Unhide the cursor.

Command documentation

change_vt(*vt: int*) → bool

Change virtual terminal to that specified

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

Return value is tuple *(success, result)*, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

get_inputs() → dict[str, list[dict[str, str]]]

Get information on all input devices.

hide_cursor() → None

Hide the cursor.

info() → dict[str, Any]

Get basic information about the running backend.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

query_tree() → list[int]

Get IDs of all mapped windows in ascending Z order.

set_keymap(*layout: str | None = None*, *options: str | None = None*, *variant: str | None = None*) → None

Set the keymap for the current keyboard.

The options correspond to xkbcommon configuration environmental variables and if not specified are taken from the environment. Acceptable values are strings identical to those accepted by the env variables.

unhide_cursor() → None

Unhide the cursor.

CHAPTER

SIXTEEN

HACKING ON QTILE

16.1 Requirements

Here are Qtile's additional dependencies that may be required for tests:

Dependency	Ubuntu Package	Needed for
pytest	python3-pytest	Running tests
pre-commit	pre-commit	Running linters
PyGObject	python3-gi	Running tests (test windows)
Xephyr	xserver-xephyr	Testing with X11 backend (optional, see below)
mypy	python3-mypy	Testing `qtile check` (optional)
imagemagick>=6.8	imagemagick	`test/test_images*` (optional)
gtk-layer-shell	libgtk-layer-shell0	Testing notification windows in Wayland (optional)
dbus-launch	dbus-x11	Testing dbus-using widgets (optional)
notifiy-send	libnotify-bin	Testing `Notify` widget (optional)
xvfb	xvfb	Testing with X11 headless (optional)

16.1.1 Backends

The test suite can be run using the X11 or Wayland backend, or both. By default, only the X11 backend is used for tests. To test a single backend or both backends, specify as arguments to pytest:

```
pytest --backend wayland  # Test just Wayland backend
pytest --backend x11 --backend wayland  # Test both
```

Testing with the X11 backend requires Xephyr (and xvfb for headless mode) in addition to the core dependencies.

16.2 Building cffi module

Qtile ships with a small in-tree pangocairo binding built using cffi, `pangocffi.py`, and also binds to xcursor with cffi. The bindings are not built at run time and will have to be generated manually when the code is downloaded or when any changes are made to the cffi library. This can be done by calling:

```
./scripts/ffibuild
```

16.3 Setting up the environment

In the root of the project, run `./dev.sh`. It will create a virtualenv called `venv`.

Activate this virtualenv with `. venv/bin/activate`. Deactivate it with the `deactivate` command.

16.4 Building the documentation

To build the documentation, you will also need to install graphviz.

Go into the `docs/` directory and run `pip install -r requirements.txt`.

Build the documentation with `make html`.

Check the result by opening `_build/html/index.html` in your browser.

Note: To speed up local testing, screenshots are not generated each time the documentation is built.

You can enable screenshots by setting the `QTILE_BUILD_SCREENSHOTS` environmental variable at build time e.g. `QTILE_BUILD_SCREENSHOTS=1 make html`. You can also export the variable so it will apply to all local builds `export QTILE_BUILD_SCREENSHOTS=1` (but remember to unset it if you want to skip building screenshots).

16.5 Development and testing

In practice, the development cycle looks something like this:

1. make minor code change
2. run appropriate test: `pytest tests/test_module.py` or `pytest -k PATTERN`
3. GOTO 1, until hackage is complete
4. run entire test suite to make sure you didn't break anything else: `pytest`
5. try to commit, get changes and feedback from the pre-commit hooks
6. GOTO 5, until your changes actually get committed

Tests and pre-commit hooks will be run by our CI on every pull request as well so you can see whether or not your contribution passes.

16.6 Coding style

While not all of our code follows PEP8, we do try to adhere to it where possible. All new code should be PEP8 compliant.

The `make lint` command (or `pre-commit run -a`) will run our linters and formatters with our configuration over the whole libqtile to ensure your patch complies with reasonable formatting constraints. We also request that git commit messages follow the standard format.

16.7 Logging

Logs are important to us because they are our best way to see what Qtile is doing when something abnormal happens. However, our goal is not to have as many logs as possible, as this hinders readability. What we want are relevant logs.

To decide which log level to use, refer to the following scenarios:

- ERROR: a problem affects the behavior of Qtile in a way that is noticeable to the end user, and we can't work around it.
- WARNING: a problem causes Qtile to operate in a suboptimal manner.
- INFO: the state of Qtile has changed.
- DEBUG: information is worth giving to help the developer better understand which branch the process is in.

Be careful not to overuse DEBUG and clutter the logs. No information should be duplicated between two messages.

Also, keep in mind that any other level than DEBUG is aimed at users who don't necessarily have advanced programming knowledge; adapt your message accordingly. If it can't make sense to your grandma, it's probably meant to be a DEBUG message.

16.8 Deprecation policy

When a widget API is changed, you should deprecate the change using `libqtile.widget.base.deprecated` to warn users, in addition to adding it to the appropriate place in the changelog. We will typically remove deprecated APIs one tag after they are deprecated.

16.9 Using Xephyr

Qtile has a very extensive test suite, using the Xephyr nested X server. When tests are run, a nested X server with a nested instance of Qtile is fired up, and then tests interact with the Qtile instance through the client API. The fact that we can do this is a great demonstration of just how completely scriptable Qtile is. In fact, Qtile is designed expressly to be scriptable enough to allow unit testing in a nested environment.

The Qtile repo includes a tiny helper script to let you quickly pull up a nested instance of Qtile in Xephyr, using your current configuration. Run it from the top-level of the repository, like this:

```
./scripts/xephyr
```

Change the screen size by setting the `SCREEN_SIZE` environment variable. Default: 800x600. Example:

```
SCREEN_SIZE=1920x1080 ./scripts/xephyr
```

Change the log level by setting the `LOG_LEVEL` environment variable. Default: INFO. Example:

```
LOG_LEVEL=DEBUG ./scripts/xephyr
```

The script will also pass any additional options to Qtile. For example, you can use a specific configuration file like this:

```
./scripts/xephyr -c ~/.config/qtile/other_config.py
```

Once the Xephyr window is running and focused, you can enable capturing the keyboard shortcuts by hitting Control+Shift. Hitting them again will disable the capture and let you use your personal keyboard shortcuts again.

You can close the Xephyr window by enabling the capture of keyboard shortcuts and hit Mod4+Control+Q. Mod4 (or Mod) is usually the Super key (or Windows key). You can also close the Xephyr window by running `qtile cmd-obj -o cmd -f shutdown` in a terminal (from inside the Xephyr window of course).

You don't need to run the Xephyr script in order to run the tests as the test runner will launch its own Xephyr instances.

16.10 Second X Session

Some users prefer to test Qtile in a second, completely separate X session: Just switch to a new tty and run `startx` normally to use the `~/.xinitrc` X startup script.

It's likely though that you want to use a different, customized startup script for testing purposes, for example `~/.config/qtile/xinitrc`. You can do so by launching X with:

```
startx ~/.config/qtile/xinitrc
```

`startx` deals with multiple X sessions automatically. If you want to use `xinit` instead, you need to first copy `/etc/X11/xinit/xserverrc` to `~/.xserverrc`; when launching it, you have to specify a new session number:

```
xinit ~/.config/qtile/xinitrc -- :1
```

Examples of custom X startup scripts are available in qtile-examples.

16.11 Debugging in PyCharm

Make sure to have all the requirements installed and your development environment setup.

PyCharm should automatically detect the `venv` virtualenv when opening the project. If you are using another viirtualenv, just instruct PyCharm to use it in `Settings -> Project: qtile -> Project interpreter`.

In the project tree, on the left, right-click on the `libqtile` folder, and click on `Mark Directory as -> Sources Root`.

Next, add a Configuration using a Python template with these fields:

- Script path: `bin/qtile`, or the absolute path to it
- Parameters: `-c libqtile/resources/default_config.py`, or nothing if you want to use your own config file in `~/.config/qtile/config.py`
- Environment variables: `PYTHONUNBUFFERED=1;DISPLAY=:1`
- Working directory: the root of the project
- Add contents root to PYTHONPATH: yes
- Add source root to PYTHONPATH: yes

Then, in a terminal, run:

    Xephyr +extension RANDR -screen 1920x1040 :1 -ac &

Note that we used the same display, `:1`, in both the terminal command and the PyCharm configuration environment variables. Feel free to change the screen size to fit your own screen.

Finally, place your breakpoints in the code and click on `Debug`!

Once you finished debugging, you can close the Xephyr window with `kill PID` (use the `jobs` builtin to get its PID).

16.12 Debugging in VSCode

Make sure to have all the requirements installed and your development environment setup.

Open the root of the repo in VSCode. If you have created it, VSCode should detect the `venv` virtualenv, if not, select it.

Create a launch.json file with the following lines.

```
{
  "version": "0.2.0",
  "configurations": [
      {
          "name": "Python: Qtile",
          "type": "python",
          "request": "launch",
          "program": "${workspaceFolder}/bin/qtile",
          "cwd": "${workspaceFolder}",
          "args": ["-c", "libqtile/resources/default_config.py"],
          "console": "integratedTerminal",
          "env": {"PYTHONUNBUFFERED":"1", "DISPLAY":":1"}
      }
  ]
}
```

Then, in a terminal, run:

```
Xephyr +extension RANDR -screen 1920x1040 :1 -ac &
```

Note that we used the same display, `:1`, in both the terminal command and the VSCode configuration environment variables. Then `debug` usually in VSCode. Feel free to change the screen size to fit your own screen.

16.13 Resources

Here are a number of resources that may come in handy:

- Inter-Client Conventions Manual
- Extended Window Manager Hints
- A reasonable basic Xlib Manual

CHAPTER

SEVENTEEN

CONTRIBUTING

17.1 Reporting bugs

Perhaps the easiest way to contribute to Qtile is to report any bugs you run into on the GitHub issue tracker.

Useful bug reports are ones that get bugs fixed. A useful bug report normally has two qualities:

1. **Reproducible.** If your bug is not reproducible it will never get fixed. You should clearly mention the steps to reproduce the bug. Do not assume or skip any reproducing step. Describe the issue, step-by-step, so that it is easy to reproduce and fix.
2. **Specific.** Do not write an essay about the problem. Be specific and to the point. Try to summarize the problem in a succinct manner. Do not combine multiple problems even if they seem to be similar. Write different reports for each problem.

Ensure to include any appropriate log entries from `~/.local/share/qtile/qtile.log` and/or `~/.xsession-errors`! Sometimes, an `xtrace` is requested. If that is the case, refer to *capturing an xtrace*.

17.2 Writing code

To get started writing code for Qtile, check out our guide to *Hacking on Qtile*. A more detailed page on creating widgets is available *here*.

Important: Use a separate **git branch** to make rebasing easy. Ideally, you would `git checkout -b <my_feature_branch_name>` before starting your work.

See also: *using git*.

17.2.1 Submit a pull request

You've done your hacking and are ready to submit your patch to Qtile. Great! Now it's time to submit a pull request to our issue tracker on GitHub.

Important: Pull requests are not considered complete until they include all of the following:

- **Code** that conforms to our linters and formatters. Run `pre-commit install` to install pre-commit hooks that will make sure your code is compliant before any commit.
- **Unit tests** that pass locally and in our CI environment (More below). *Please add unit tests* to ensure that your code works and stays working!

- **Documentation** updates on an as needed basis.
- A `qtile migrate` **migration** is required for config-breaking changes. See *here* for current migrations and see below for further information.
- **Code** that does not include *unrelated changes*. Examples for this are formatting changes, replacing quotes or whitespace in other parts of the code or "fixing" linter warnings popping up in your editor on existing code. *Do not include anything like the above!*
- **Widgets** don't need to catch their own exceptions, or introduce their own polling infrastructure. The code in `libqtile.widget.base.*` does all of this. Your widget should generally only include whatever parsing/rendering code is necessary, any other changes should go at the framework level. Make sure to double-check that you are not re-implementing parts of `libqtile.widget.base`.
- **Commit messages** are more important that Github PR notes, since this is what people see when they are spelunking via `git blame`. Please include all relevant detail in the actual git commit message (things like exact stack traces, copy/pastes of discussion in IRC/mailing lists, links to specifications or other API docs are all good). If your PR fixes a Github issue, it might also be wise to link to it with `#1234` in the commit message.
- PRs with **multiple commits** should not introduce code in one patch to then change it in a later patch. Please do a patch-by-patch review of your PR, and make sure each commit passes CI and makes logical sense on its own. In other words: *do* introduce your feature in one commit and maybe add the tests and documentation in a seperate commit. *Don't* push commits that partially implement a feature and are basically broken.

Note: Others might ban *force-pushes*, we allow them and prefer them over incomplete commits or commits that have a bad and meaningless commit description.

Feel free to add your contribution (no matter how small) to the appropriate place in the CHANGELOG as well!

17.2.2 Unit testing

We must test each *unit* of code to ensure that new changes to the code do not break existing functionality. The framework we use to test Qtile is pytest. How pytest works is outside of the scope of this documentation, but there are tutorials online that explain how it is used.

Our tests are written inside the `test` folder at the top level of the repository. Reading through these, you can get a feel for the approach we take to test a given unit. Most of the tests involve an object called `manager`. This is the test manager (defined in test/helpers.py), which exposes a command client at `manager.c` that we use to test a Qtile instance running in a separate thread as if we were using a command client from within a running Qtile session.

For any Qtile-specific question on testing, feel free to ask on our issue tracker or on IRC (#qtile on irc.oftc.net).

17.2.3 Running tests locally

This section gives an overview about `tox` so that you don't have to search its documentation just to get started.

Checks are grouped in so-called `environments`. Some of them are configured to check that the code works (the usual unit test, e.g. `py39`, `pypy3`), others make sure that your code conforms to the style guide (`pep8`, `codestyle`, `mypy`). A third kind of test verifies that the documentation and packaging processes work (`docs`, `docstyle`, `packaging`).

We have configured `tox` to run the full suite of tests whenever a pull request is submitted/updated. To reduce the amount of time taken by these tests, we have created separate environments for both python versions and backends (e.g. tests for x11 and wayland run in parallel for each python version that we currently support).

These environments were designed with automation in mind so there are separate `test` environments which should be used for running qtile's tests locally. By default, tests will only run on x11 backend (but see below for information on how to set the backend).

The following examples show how to run tests locally:

- To run the functional tests, use `tox -e test`. You can specify to only run a specific test file or even a specific test within that file with the following commands:

```
tox -e test # Run all tests in default python version
tox -e test -- -x test/widgets/test_widgetbox.py  # run a single file
tox -e test -- -x test/widgets/test_widgetbox.py::test_widgetbox_widget
tox -e test -- --backend=wayland --backend=x11  # run tests on both backends
tox -e test-both  # same as above
tox -e test-wayland  # Just run tests on wayland backend
```

- To run style and building checks, use `tox -e docs,packaging,pep8,...`. You can use `-p auto` to run the environments in parallel.

Important: The CI is configured to run all the environments. Hence it can be time- consuming to make all the tests pass. As stated above, pull requests that don't pass the tests are considered incomplete. Don't forget that this does not only include the functionality, but the style, typing annotations (if necessary) and documentation as well!

17.2.4 Writing migrations

Migrations are needed when a commit introduces a change which makes a breaking change to a user's config. Examples include renaming classes, methods, arguments and moving modules or class definitions.

Where these changes are made, it is strongly encouraged to support the old syntax where possible and warn the user about the deprecations.

Whether or not a deprecation warning is provided, a migration script should be provided that will modify the user's config when they run `qtile migrate`.

Click here for detailed instructions on *How to write a migration script*.

How to write a migration script

Qtile's migration scripts should provide two functions:

- Update config files to fix any breaking changes introduced by a commit
- Provide linting summary of errors in existing configs

To do this, we use LibCST to parse the config file and make changes as appropriate. Basic tips for using `LibCST` are included below but it is recommended that you read their documentation to familiarise yourself with the available functionalities.

Stucture of a migration file

Migrations should be saved as a new file in `libqtile/scripts/migrations`.

A basic migration will look like this:

```
from libqtile.scripts.migrations._base import MigrationTransformer, _QtileMigrator, add_
↪migration


class MyMigration(MigrationTransformer):
    """The class that actually modifies the code."""
    ...


class Migrator(_QtileMigrator):
    ID = "MyMigrationName"

    SUMMARY = "Summary of migration."

    HELP = """
    Longer text explaining purpose of the migration and, ideally,
    giving code examples.

    """

    AFTER_VERSION = "0.22.1"

    TESTS = []

    visitor = MyMigration


add_migration(Migrator)
```

Providing details about the migration

The purpose of `Migrator` class in the code above is to provide the information about the migration.

It is important that the information is as helpful as possible as it is used in multiple places.

- The `ID` attribute is a short, unique name to identify the migration. This allows users to select specific migrations to run via `qtile migrate --run-migrations ID`.
- The `SUMMARY` attribute is used to provide a brief summary of the migration and is used when a user runs `qtile migrate --list-migrations`. It is also used in the documentation.
- Similarly, the `HELP` attribute is used for the script (`qtie migrate --info ID`) and the documentation. This text should be longer and can include example code. As it is used in the documentation, it should use RST syntax (e.g. `.. code:: python` for codeblocks etc.).
- `AFTER_VERSION` should be set the name of the current release. This allows users to filter migrations to those that were added after the last release.
- The `visitor` attribute is a link to the class definition (not and instance of the class) for the transformer that you wish to use.

- The `add_migration` call at the end is required to ensure the migration is loaded into the list of available migrations.
- See below for details on `TESTS`.

How migrations are run

You are pretty much free to transform the code as you see fit. By default, the script will run the `visit` method on the parsed code and will pass the `visitor` attribute of the `_QtileMigrator` class object. Therefore, if all your transformations can be performed in a single visitor, it is not necessary to do anything further in the `Migrator` class.

However, if you want to run mutiple visitors, transformers, codemods, this is possible by overriding the `run` method of the `_QtileMigrator` class. For example, the `RemoveCmdPrefix` migrator has the following code:

```
def run(self, original):
    # Run the base migrations
    transformer = CmdPrefixTransformer()
    updated = original.visit(transformer)
    self.update_lint(transformer)

    # Check if we need to add an import line
    if transformer.needs_import:
        # We use the built-in visitor to add the import
        context = codemod.CodemodContext()
        AddImportsVisitor.add_needed_import(
            context, "libqtile.command.base", "expose_command"
        )
        visitor = AddImportsVisitor(context)

        # Run the visitor over the updated code
        updated = updated.visit(visitor)

    return original, updated
```

In this migration, it may be required to add an import statement. `LibCST` has a built-in transformation for doing this so we can run that after our own transformation has been performed.

Important: The `run` method must return a tuple of the original code and the updated code.

Transforming the code

It is recommended that you use a transformed to update the code. For convenience, a `MigrationTransformer` class is defined in `libqtile.scripts.migrations._base`. This class definition includes some metadata information and a `lint` method for outputting details of errors.

Let's look at an example transformer to understand how the migration works. The code below shows how to change a positional argument to a keyword argument in the `WidgetBox` widget.

```
class WidgetboxArgsTransformer(MigrationTransformer):
    @m.call_if_inside(
        m.Call(func=m.Name("WidgetBox")) | m.Call(func=m.Attribute(attr=m.Name("WidgetBox
↪")))
```

(continues on next page)

(continued from previous page)

```
    )
    @m.leave(m.Arg(keyword=None))
    def update_widgetbox_args(self, original_node, updated_node) -> cst.Arg:
        """Changes positional  argumentto 'widgets' kwargs."""
        self.lint(
            original_node,
            "The positional argument should be replaced with a keyword argument named
↪'widgets'.",
        )
        return updated_node.with_changes(keyword=cst.Name("widgets"), equal=EQUALS_NO_
↪SPACE)
```

Our class (which inherits from `MigrationTransformer`) defines a single method to perform the transformation. We take advantage of `LibCST` and its Matchers to narrow the scope of when the transformation is run.

We are looking to modify an argument so we use the `@m.leave(m.Arg())` decorator to call the function at end of parsing an argument. We can restrict when this is called by specify `m.Arg(keyword=None)` so that it is only called for positional arguments. Furthermore, as we only want this called for `WidgetBox` instantiation lines, we add an additional decorator `@m.call_if_inside(m.Call())`. This ensures the method is only called when we're in a call. On its own, that's not helpful as args would almost always be part of a call. However, we can say we only want to match calls to `WidgetBox`. The reason for the long syntax above is that `LibCST` parses `WidgetBox()` and `widget.WidgetBox()` differently. In the first one, `WidgetBox` is in the `func` property of the call. However, in the second, the `func` is an `Attribute` as it is a dotted name and so we need to check the `attr` property.

The decorated method takes two arguments, `original_mode` and `updated_node` (note: The `original_node` should not be modified). The method should also confirm the return type.

The above method provides a linting message by calling `self.lint` and passing the original node and a helpful message.

Finally, the method updates the code by calling `updated_node.with_changes()`. In this instance, we add a keyword (`"widgets"`) to the argument. We also remove spaces around the equals sign as these are added by default by `LibCST`. The updated node is returned.

Helper classes

Helper classes are provided for common transformations.

- `RenamerTransformer` will update all instances of a name, replacing it with another. The class will also handle the necessary linting.

```
class RenameHookTransformer(RenamerTransformer):
    from_to = ("window_name_change", "client_name_updated")
```

Testing the migration

All migrations must be tested, ideally with a number of scenarios to confirm that the migration works as expected.

Unlike other tests, the tests for the migrations are defined within the `TESTS` attribute.

This is a list that should take a `Check`, `Change` or `NoChange` object (all are imported from `libqtile.scripts.migrations._base`).

A `Change` object needs two parameters, the input code and the expected output. A `NoChange` object just defines the input (as the output should be the same).

A `Check` object is identical to `Change` however, when running the test suite, the migrated code will be verified with `qtile check`. The code will therefore need to include all relevant imports etc.

Based on the above, the following is recommended as best practice:

- Define one `Check` test which addresses every situation anticipated by the migration
- Use as many `Change` tests as required to test individual scenarios in a minimal way
- Use `NoChange` tests where there are specific cases that should not be modified
- Depending on the simplicity of the migration, a single `Check` may be all that is required

For example, the `RemoveCmdPrefix` migration has the following `TESTS`:

```
TESTS = [
    Change("""qtile.cmd_spawn("alacritty")""", """qtile.spawn("alacritty")"""),
    Change("""qtile.cmd_groups()""", """qtile.get_groups()"""),
    Change("""qtile.cmd_screens()""", """qtile.get_screens()"""),
    Change("""qtile.current_window.cmd_hints()""", """qtile.current_window.get_hints()""
↪"),
    Change(
        """qtile.current_window.cmd_opacity(0.5)""",
        """qtile.current_window.set_opacity(0.5)""",
    ),
    Change(
        """
        class MyWidget(widget.Clock):
            def cmd_my_command(self):
                pass
        """,
        """
        from libqtile.command.base import expose_command

        class MyWidget(widget.Clock):
            @expose_command
            def my_command(self):
                pass
        """
    ),
    NoChange(
        """
        def cmd_some_other_func():
            pass
        """
    ),
```

(continues on next page)

(continued from previous page)

```
    Check(
        """
        from libqtile import qtile, widget

        class MyClock(widget.Clock):
            def cmd_my_exposed_command(self):
                pass

        def my_func(qtile):
            qtile.cmd_spawn("rickroll")
            hints = qtile.current_window.cmd_hints()
            groups = qtile.cmd_groups()
            screens = qtile.cmd_screens()
            qtile.current_window.cmd_opacity(0.5)

        def cmd_some_other_func():
            pass
        """,
        """
        from libqtile import qtile, widget
        from libqtile.command.base import expose_command

        class MyClock(widget.Clock):
            @expose_command
            def my_exposed_command(self):
                pass

        def my_func(qtile):
            qtile.spawn("rickroll")
            hints = qtile.current_window.get_hints()
            groups = qtile.get_groups()
            screens = qtile.get_screens()
            qtile.current_window.set_opacity(0.5)

        def cmd_some_other_func():
            pass
        """
    )
]
```

The tests check:

- `cmd_` prefix is removed on method calls, updating specific changes as required
- Exposed methods in a class should use the `expose_command` decorator (adding the import if it's not already included)
- No change is made to a function definition (as it's not part of a class definition)

Note: Tests will fail in the following scenarios:

- If no tests are defined
- If a `Change` test does not result in linting output

- If no `Check` test is defined

You can check your tests by running `pytest -k <YourMigrationID>`. Note, `mpypy` must be installed for the `Check` tests to be run.

CHAPTER

EIGHTEEN

FREQUENTLY ASKED QUESTIONS

18.1 Why the name Qtile?

Users often wonder, why the Q? Does it have something to do with Qt? No. Below is an IRC excerpt where cortesi explains the great trial that ultimately brought Qtile into existence, thanks to the benevolence of the Open Source Gods. Praise be to the OSG!

```
ramnes:  what does Qtile mean?
ramnes:  what's the Q?
@tych0:  ramnes: it doesn't :)
@tych0:  cortesi was just looking for the first letter that wasn't registered
         in a domain name with "tile" as a suffix
@tych0:  qtile it was :)
cortesi: tych0, dx: we really should have something more compelling to
         explain the name. one day i was swimming at manly beach in sydney,
         where i lived at the time. suddenly, i saw an enormous great white
         right beside me. it went for my leg with massive, gaping jaws, but
         quick as a flash, i thumb-punched it in both eyes. when it reared
         back in agony, i saw that it had a jagged, gnarly scar on its
         stomach... a scar shaped like the letter "Q".
cortesi: while it was distracted, i surfed a wave to shore. i knew that i
         had to dedicate my next open source project to the ocean gods, in
         thanks for my lucky escape. and thus, qtile got its name...
```

18.2 When I first start xterm/urxvt/rxvt containing an instance of Vim, I see text and layout corruption. What gives?

Vim is not handling terminal resizes correctly. You can fix the problem by starting your xterm with the "-wf" option, like so:

```
xterm -wf -e vim
```

Alternatively, you can just cycle through your layouts a few times, which usually seems to fix it.

18.3 How do I know which modifier specification maps to which key?

To see a list of modifier names and their matching keys, use the `xmodmap` command. On my system, the output looks like this:

```
$ xmodmap
xmodmap:  up to 3 keys per modifier, (keycodes in parentheses):

shift       Shift_L (0x32),  Shift_R (0x3e)
lock        Caps_Lock (0x9)
control     Control_L (0x25),  Control_R (0x69)
mod1        Alt_L (0x40),  Alt_R (0x6c),  Meta_L (0xcd)
mod2        Num_Lock (0x4d)
mod3
mod4        Super_L (0xce),  Hyper_L (0xcf)
mod5        ISO_Level3_Shift (0x5c),  Mode_switch (0xcb)
```

18.4 My "pointer mouse cursor" isn't the one I expect it to be!

Qtile should set the default cursor to left_ptr, you must install xcb-util-cursor if you want support for themed cursors.

18.5 LibreOffice menus don't appear or don't stay visible

A workaround for problem with the mouse in libreoffice is setting the environment variable »SAL_USE_VCLPLUGIN=gen«. It is dependent on your system configuration as to where to do this. e.g. ArchLinux with libreoffice-fresh in /etc/profile.d/libreoffice-fresh.sh.

18.6 How can I get my groups to stick to screens?

This behaviour can be replicated by configuring your keybindings to not move groups between screens. For example if you want groups `"1"`, `"2"` and `"3"` on one screen and `"q"`, `"w"`, and `"e"` on the other, instead of binding keys to `lazy.group[name].toscreen()`, use this:

```
groups = [
    # Screen affinity here is used to make
    # sure the groups startup on the right screens
    Group(name="1", screen_affinity=0),
    Group(name="2", screen_affinity=0),
    Group(name="3", screen_affinity=0),
    Group(name="q", screen_affinity=1),
    Group(name="w", screen_affinity=1),
    Group(name="e", screen_affinity=1),
]

def go_to_group(name: str):
    def _inner(qtile):
        if len(qtile.screens) == 1:
```

(continues on next page)

(continued from previous page)

```
            qtile.groups_map[name].toscreen()
            return

        if name in '123':
            qtile.focus_screen(0)
            qtile.groups_map[name].toscreen()
        else:
            qtile.focus_screen(1)
            qtile.groups_map[name].toscreen()

    return _inner

for i in groups:
    keys.append(Key([mod], i.name, lazy.function(go_to_group(i.name))))
```

To be able to move windows across these groups which switching groups, a similar function can be used:

```
def go_to_group_and_move_window(name: str):
    def _inner(qtile):
        if len(qtile.screens) == 1:
            qtile.current_window.togroup(name, switch_group=True)
            return

        if name in "123":
            qtile.current_window.togroup(name, switch_group=False)
            qtile.focus_screen(0)
            qtile.groups_map[name].toscreen()
        else:
            qtile.current_window.togroup(name, switch_group=False)
            qtile.focus_screen(1)
            qtile.groups_map[name].toscreen()

    return _inner

for i in groups:
    keys.append(Key([mod, "shift"], i.name, lazy.function(go_to_group_and_move_window(i.
↪name))))
```

If you use the `GroupBox` widget you can make it reflect this behaviour:

```
groupbox1 = widget.GroupBox(visible_groups=['1', '2', '3'])
groupbox2 = widget.GroupBox(visible_groups=['q', 'w', 'e'])
```

And if you jump between having single and double screens then modifying the visible groups on the fly may be useful:

```
@hook.subscribe.screens_reconfigured
async def _():
    if len(qtile.screens) > 1:
        groupbox1.visible_groups = ['1', '2', '3']
    else:
        groupbox1.visible_groups = ['1', '2', '3', 'q', 'w', 'e']
    if hasattr(groupbox1, 'bar'):
        groupbox1.bar.draw()
```

18.7 Where can I find example configurations and other scripts?

Please visit our qtile-examples repo which contains examples of users' configurations, scripts and other useful links.

18.8 Where are the log files for Qtile?

The log files for qtile are at `~/.local/share/qtile/qtile.log`.

18.9 How can I match the bar with picom?

You can use `"QTILE_INTERNAL:32c = 1"` in your picom.conf to match the bar. This will match all internal Qtile windows, so if you want to avoid that or to target bars individually, you can set a custom property and match that:

```
mybar = Bar(...)

@hook.subscribe.startup
def _():
    mybar.window.window.set_property("QTILE_BAR", 1, "CARDINAL", 32)
```

This would enable matching on `mybar`'s window using `"QTILE_BAR:32c = 1"`. See 2526 and 1515 for more discussion.

18.10 Why do get a warning that fonts cannot be loaded?

When installing Qtile on a new system, when running the test suite or the Xephyr script (`./scripts/xephyr`), you might see errors in the output like the following or similar:

- Xephyr script:

```
xterm: cannot load font "-Misc-Fixed-medium-R-*-*-13-120-75-75-C-120-ISO10646-1"
xterm: cannot load font "-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-
↪iso10646-1"
```

- `pytest`:

```
---------- Captured stderr call ----------
Warning: Cannot convert string "8x13" to type FontStruct
Warning: Unable to load any usable ISO8859 font
Warning: Unable to load any usable ISO8859 font
Error: Aborting: no font found

-------- Captured stderr teardown --------
Qtile exited with exitcode: -9
```

If it happens, it might be because you're missing fonts on your system.

On ArchLinux, you can fix this by installing `xorg-fonts-misc`:

```
sudo pacman -S xorg-fonts-misc
```

Try to search for "xorg fonts misc" with your distribution name on the internet to find how to install them.

18.11 I've upgraded and Qtile's broken. What do I do?

If you've recently upgraded, the first thing to do is check the *changelog* and see if any breaking changes were made.

Next, check your log file (see above) to see if any error messages explain what the problem is.

If you're still stuck, come and ask for help on Discord, IRC or GitHub.

CHAPTER

NINETEEN

HOW TO CREATE A WIDGET

The aim of this page is to explain the main components of qtile widgets, how they work, and how you can use them to create your own widgets.

Note: This page is not meant to be an exhaustive summary of everything needed to make a widget.

It is highly recommended that users wishing to create their own widget refer to the source documentation of existing widgets to familiarise themselves with the code.

However, the detail below may prove helpful when read in conjunction with the source code.

19.1 What is a widget?

In Qtile, a widget is a small drawing that is displayed on the user's bar. The widget can display text, images and drawings. In addition, the widget can be configured to update based on timers, hooks, dbus_events etc. and can also respond to mouse events (clicks, scrolls and hover).

19.2 Widget base classes

Qtile provides a number of base classes for widgets than can be used to implement commonly required features (e.g. display text).

Your widget should inherit one of these classes. Whichever base class you inherit for your widget, if you override either the `__init__` and/or `_configure` methods, you should make sure that your widget calls the equivalent method from the superclass.

```
class MyCustomWidget(base._TextBox):

    def __init__(self, **config):
        super().__init__("", **config)
        # My widget's initialisation code here
```

The functions of the various base classes are explained further below.

19.2.1 _Widget

This is the base widget class that defines the core components required for a widget. All other base classes are based off this class.

This is like a blank canvas so you're free to do what you want but you don't have any of the extra functionality provided by the other base classes.

The `base._Widget` class is therefore typically used for widgets that want to draw graphics on the widget as opposed to displaying text.

19.2.2 _TextBox

The `base._TextBox` class builds on the bare widget and adds a `drawer.TextLayout` which is accessible via the `self.layout` property. The widget will adjust its size to fit the amount of text to be displayed.

Text can be updated via the `self.text` property but note that this does not trigger a redrawing of the widget.

Parameters including `font`, `fontsize`, `fontshadow`, `padding` and `foreground` (font colour) can be configured. It is recommended not to hard-code these parameters as users may wish to have consistency across units.

19.2.3 InLoopPollText

The `base.InLoopPollText` class builds on the `base._TextBox` by adding a timer to periodically refresh the displayed text.

Widgets using this class should override the `poll` method to include a function that returns the required text.

Note: This loop runs in the event loop so it is important that the poll method does not call some blocking function. If this is required, widgets should inherit the `base.ThreadPoolText` class (see below).

19.2.4 ThreadPoolText

The `base.ThreadPoolText` class is very similar to the `base.InLoopPollText` class. The key difference is that the `poll` method is run asynchronously and triggers a callback once the function completes. This allows widgets to get text from long-running functions without blocking Qtile.

19.3 Mixins

As well as inheriting from one of the base classes above, widgets can also inherit one or more mixins to provide some additional functionality to the widget.

19.3.1 PaddingMixin

This provides the `padding(_x|_y|)` attributes which can be used to change the appearance of the widget.

If you use this mixin in your widget, you need to add the following line to your `__init__` method:

```
self.add_defaults(base.PaddingMixin.defaults)
```

19.3.2 MarginMixin

The `MarginMixin` is essentially effectively exactly the same as the `PaddingMixin` but, instead, it provides the `margin(_x|_y|)` attributes.

As above, if you use this mixin in your widget, you need to add the following line to your `__init__` method:

```
self.add_defaults(base.MarginMixin.defaults)
```

19.4 Configuration

Now you know which class to base your widget on, you need to know how the widget gets configured.

19.4.1 Defining Parameters

Each widget will likely have a number of parameters that users can change to customise the look and feel and/or behaviour of the widget for their own needs.

The widget should therefore provide the default values of these parameters as a class attribute called `defaults`. The format of this attribute is a list of tuples.

```
defaults = [
    ("parameter_name",
     default_parameter_value,
     "Short text explaining what parameter does")
]
```

Users can override the default value when creating their `config.py` file.

```
MyCustomWidget(parameter_name=updated_value)
```

Once the widget is initialised, these parameters are available at `self.parameter_name`.

19.4.2 The __init__ method

Parameters that should not be changed by users can be defined in the `__init__` method.

This method is run when the widgets are initially created. This happens before the `qtile` object is available.

19.4.3 The _configure method

The `_configure` method is called by the `bar` object and sets the `self.bar` and `self.qtile` attributes of the widget. It also creates the `self.drawer` attribute which is necessary for displaying any content.

Once this method has been run, your widget should be ready to display content as the bar will draw once it has finished its configuration.

Calls to methods required to prepare the content for your widget should therefore be made from this method rather than `__init__`.

19.5 Displaying output

A Qtile widget is just a drawing that is displayed at a certain location the user's bar. The widget's job is therefore to create a small drawing surface that can be placed in the appropriate location on the bar.

19.5.1 The "draw" method

The `draw` method is called when the widget needs to update its appearance. This can be triggered by the widget itself (e.g. if the content has changed) or by the bar (e.g. if the bar needs to redraw its entire contents).

This method therefore needs to contain all the relevant code to draw the various components that make up the widget. Examples of displaying text, icons and drawings are set out below.

It is important to note that the bar controls the placing of the widget by assigning the `offsetx` value (for horizontal positioning) and `offsety` value (for vertical positioning). Widgets should use this at the end of the `draw` method. Both `offsetx` and `offsety` are required as both values will be set if the bar is drawing a border.

```
self.drawer.draw(offsetx=self.offsetx, offsety=self.offsety, width=self.width)
```

Note: If you need to trigger a redrawing of your widget, you should call `self.draw()` if the width of your widget is unchanged. Otherwise you need to call `self.bar.draw()` as this method means the bar recalculates the position of all widgets.

19.5.2 Displaying text

Text is displayed by using a `drawer.TextLayout` object. If all you are doing is displaying text then it's highly recommended that you use the `base._TextBox` superclass as this simplifies adding and updating text.

If you wish to implement this manually then you can create a your own `drawer.TextLayout` by using the `self.drawer.textlayout` method of the widget (only available after the *_configure* method has been run). object to include in your widget.

Some additional formatting of Text can be displayed using pango markup and ensuring the `markup` parameter is set to `True`.

```
self.textlayout = self.drawer.textlayout(
                    "Text",
                    "fffff",       # Font colour
                    "sans",        # Font family
                    12,            # Font size
```

(continues on next page)

(continued from previous page)

```
                    None,           # Font shadow
                    markup=False,   # Pango markup (False by default)
                    wrap=True       # Wrap long lines (True by default)
                    )
```

19.5.3 Displaying icons and images

Qtile provides a helper library to convert images to a `surface` that can be drawn by the widget. If the images are static then you should only load them once when the widget is configured. Given the small size of the bar, this is most commonly used to draw icons but the same method applies to other images.

```
from libqtile import images

def setup_images(self):

    self.surfaces = {}

    # File names to load (will become keys to the `surfaces` dictionary)
    names = (
        "audio-volume-muted",
        "audio-volume-low",
        "audio-volume-medium",
        "audio-volume-high"
    )

    d_images = images.Loader(self.imagefolder)(*names)  # images.Loader can take more␣
↪than one folder as an argument

    for name, img in d_images.items():
        new_height = self.bar.height - 1
        img.resize(height=new_height)   # Resize images to fit widget
        self.surfaces[name] = img.pattern  # Images added to the `surfaces` dictionary
```

Drawing the image is then just a matter of painting it to the relevant surface:

```
def draw(self):
    self.drawer.ctx.set_source(self.surfaces[img_name])  # Use correct key here for your␣
↪image
    self.drawer.ctx.paint()
    self.drawer.draw(offsetx=self.offset, width=self.length)
```

19.5.4 Drawing shapes

It is possible to draw shapes directly to the widget. The `Drawer` class (available in your widget after configuration as `self.drawer`) provides some basic functions `rounded_rectangle`, `rounded_fillrect`, `rectangle` and `fillrect`.

In addition, you can access the Cairo context drawing functions via `self.drawer.ctx`.

For example, the following code can draw a wifi icon showing signal strength:

```
import math

...

def to_rads(self, degrees):
    return degrees * math.pi / 180.0

def draw_wifi(self, percentage):

    WIFI_HEIGHT = 12
    WIFI_ARC_DEGREES = 90

    y_margin = (self.bar.height - WIFI_HEIGHT) / 2
    half_arc = WIFI_ARC_DEGREES / 2

    # Draw grey background
    self.drawer.ctx.new_sub_path()
    self.drawer.ctx.move_to(WIFI_HEIGHT, y_margin + WIFI_HEIGHT)
    self.drawer.ctx.arc(WIFI_HEIGHT,
                        y_margin + WIFI_HEIGHT,
                        WIFI_HEIGHT,
                        self.to_rads(270 - half_arc),
                        self.to_rads(270 + half_arc))
    self.drawer.set_source_rgb("666666")
    self.drawer.ctx.fill()

    # Draw white section to represent signal strength
    self.drawer.ctx.new_sub_path()
    self.drawer.ctx.move_to(WIFI_HEIGHT, y_margin + WIFI_HEIGHT)
    self.drawer.ctx.arc(WIFI_HEIGHT
                        y_margin + WIFI_HEIGHT,
                        WIFI_HEIGHT * percentage,
                        self.to_rads(270 - half_arc),
                        self.to_rads(270 + half_arc))
    self.drawer.set_source_rgb("ffffff")
    self.drawer.ctx.fill()
```

This creates something looking like this: .

19.5.5 Background

At the start of the `draw` method, the widget should clear the drawer by drawing the background. Usually this is done by including the following line at the start of the method:

```
self.drawer.clear(self.background or self.bar.background)
```

The background can be a single colour or a list of colours which will result in a linear gradient from top to bottom.

19.6 Updating the widget

Widgets will usually need to update their content periodically. There are numerous ways that this can be done. Some of the most common ones are summarised below.

19.6.1 Timers

A non-blocking timer can be called by using the `self.timeout_add` method.

```
self.timeout_add(delay_in_seconds, method_to_call, (method_args))
```

Note: Consider using the `ThreadPoolText` superclass where you are calling a function repeatedly and displaying its output as text.

19.6.2 Hooks

Qtile has a number of hooks built in which are triggered on certain events.

The `WindowCount` widget is a good example of using hooks to trigger updates. It includes the following method which is run when the widget is configured:

```
from libqtile import hook

...

def _setup_hooks(self):
    hook.subscribe.client_killed(self._win_killed)
    hook.subscribe.client_managed(self._wincount)
    hook.subscribe.current_screen_change(self._wincount)
    hook.subscribe.setgroup(self._wincount)
```

Read the *Built-in Hooks* page for details of which hooks are available and which arguments are passed to the callback function.

19.6.3 Using dbus

Qtile uses `dbus-next` for interacting with dbus.

If you just want to listen for signals then Qtile provides a helper method called `add_signal_receiver` which can subscribe to a signal and trigger a callback whenever that signal is broadcast.

Note: Qtile uses the `asyncio` based functions of `dbus-next` so your widget must make sure, where necessary, calls to dbus are made via coroutines.

There is a `_config_async` coroutine in the base widget class which can be overridden to provide an entry point for asyncio calls in your widget.

For example, the Mpris2 widget uses the following code:

```
from libqtile.utils import add_signal_receiver

...

async def _config_async(self):
    subscribe = await add_signal_receiver(
                    self.message,  # Callback function
                    session_bus=True,
                    signal_name="PropertiesChanged",
                    bus_name=self.objname,
                    path="/org/mpris/MediaPlayer2",
                    dbus_interface="org.freedesktop.DBus.Properties")
```

`dbus-next` can also be used to query properties, call methods etc. on dbus interfaces. Refer to the dbus-next documentation for more information on how to use the module.

19.7 Mouse events

By default, widgets handle button presses and will call any function that is bound to the button in the `mouse_callbacks` dictionary. The dictionary keys are as follows:

- `Button1`: Left click
- `Button2`: Middle click
- `Button3`: Right click
- `Button4`: Scroll up
- `Button5`: Scroll down
- `Button6`: Scroll left
- `Button7`: Scroll right

You can then define your button bindings in your widget (e.g. in `__init__`):

```
class MyWidget(widget.TextBox)

    def __init__(self, *args, **config):
        widget.TextBox.__init__(self, *args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
        self.add_callbacks(
            {
                "Button1": self.left_click_method,
                "Button3": self.right_click_method
            }
        )
```

Note: As well as functions, you can also bind `LazyCall` objects to button presses. For example:

```
self.add_callbacks(
    {
        "Button1": lazy.spawn("xterm"),
    }
)
```

In addition to button presses, you can also respond to mouse enter and leave events. For example, to make a clock show a longer date when you put your mouse over it, you can do the following:

```
class MouseOverClock(widget.Clock):
    defaults = [
        (
            "long_format",
            "%A %d %B %Y | %H:%M",
            "Format to show when mouse is over widget."
        )
    ]

    def __init__(self, **config):
        widget.Clock.__init__(self, **config)
        self.add_defaults(MouseOverClock.defaults)
        self.short_format = self.format

    def mouse_enter(self, *args, **kwargs):
        self.format = self.long_format
        self.bar.draw()

    def mouse_leave(self, *args, **kwargs):
        self.format = self.short_format
        self.bar.draw()
```

19.8 Exposing commands to the IPC interface

If you want to control your widget via `lazy` or scripting commands (such as `qtile cmd-obj`), you will need to expose the relevant methods in your widget. Exposing commands is done by adding the `@expose_command()` decorator to your method. For example:

```
from libqtile.command.base import expose_command
from libqtile.widget import TextBox
```

(continues on next page)

(continued from previous page)

```
class ExposedWidget(TextBox):

    @expose_command()
    def uppercase(self):
        self.update(self.text.upper())
```

Text in the `ExposedWidget` can now be made into upper case by calling `lazy.widget["exposedwidget"].uppercase()` or `qtile cmd-onj -o widget exposedwidget -f uppercase`.

If you want to expose a method under multiple names, you can pass these additional names to the decorator. For example, decorating a method with:

```
@expose_command(["extra", "additional"])
def mymethod(self):
    ...
```

will make make the method visible under `mymethod`, `extra` and `additional`.

19.9 Debugging

You can use the `logger` object to record messages in the Qtile log file to help debug your development.

```
from libqtile.log_utils import logger

...

logger.debug("Callback function triggered")
```

Note: The default log level for the Qtile log is `INFO` so you may either want to change this when debugging or use `logger.info` instead.

Debugging messages should be removed from your code before submitting pull requests.

19.10 Submitting the widget to the official repo

The following sections are only relevant for users who wish for their widgets to be submitted as a PR for inclusion in the main Qtile repo.

19.10.1 Including the widget in libqtile.widget

You should include your widget in the `widgets` dict in `libqtile.widget.__init__.py`. The relevant format is `{"ClassName": "modulename"}`.

This has a number of benefits:

- Lazy imports
- Graceful handling of import errors (useful where widget relies on third party modules)
- Inclusion in basic unit testing (see below)

19.10.2 Testing

Any new widgets should include an accompanying unit test.

Basic initialisation and configurations (using defaults) will automatically be tested by `test/widgets/test_widget_init_configure.py` if the widget has been included in `libqtile.widget.__init__.py` (see above).

However, where possible, it is strongly encouraged that widgets include additional unit tests that test specific functionality of the widget (e.g. reaction to hooks).

See *Unit testing* for more.

19.10.3 Documentation

It is really important that we maintain good documentation for Qtile. Any new widgets must therefore include sufficient documentation in order for users to understand how to use/configure the widget.

The majority of the documentation is generated automatically from your module. The widget's docstring will be used as the description of the widget. Any parameters defined in the widget's `defaults` attribute will also be displayed. It is essential that there is a clear explanation of each new parameter defined by the widget.

Screenshots

While not essential, it is strongly recommended that the documentation includes one or more screenshots.

Screenshots can be generated automatically with a minimal amount of coding by using the fixtures created by Qtile's test suite.

A screenshot file must satisfy the following criteria:

- Be named `ss_[widgetname].py`
- Any function that takes a screenshot must be prefixed with `ss_`
- Define a pytest fixture named `widget`

An example screenshot file is below:

```
import pytest

from libqtile.widget import wttr

RESPONSE = "London: +17°C"
```

(continues on next page)

(continued from previous page)

```
@pytest.fixture
def widget(monkeypatch):
    def result(self):
        return RESPONSE

    monkeypatch.setattr("libqtile.widget.wttr.Wttr.fetch", result)
    yield wttr.Wttr


@pytest.mark.parametrize(
    "screenshot_manager",
    [
        {"location": {"London": "Home"}}
    ],
    indirect=True
)
def ss_wttr(screenshot_manager):
    screenshot_manager.take_screenshot()
```

The `widget` fixture returns the widget class (not an instance of the widget). Any monkeypatching of the widget should be included in this fixture.

The screenshot function (here, called `ss_wttr`) must take an argument called `screenshot_manager`. The function can also be parameterized, in which case, each dict object will be used to configure the widget for the screenshot (and the configuration will be displayed in the docs). If you want to include parameterizations but also want to show the default configuration, you should include an empty dict (`{}`) as the first object in the list.

Taking a screenshot is then as simple as calling `screenshot_manager.take_screenshot()`. The method can be called multiple times in the same function.

`screenshot_manager.take_screenshot()` only takes a picture of the widget. If you need to take a screenshot of the bar then you need a few extra steps:

```
def ss_bar_screenshot(screenshot_manager):
    # Generate a filename for the screenshot
    target = screenshot_manager.target()

    # Get the bar object
    bar = screenshot_manager.c.bar["top"]

    # Take a screenshot. Will take screenshot of whole bar unless
    # a `width` parameter is set.
    bar.take_screenshot(target, width=width)
```

19.11 Getting help

If you still need help with developing your widget then please submit a question in the qtile-dev group or submit an issue on the github page if you believe there's an error in the codebase.

CHAPTER

TWENTY

HOW TO CREATE A LAYOUT

The aim of this page is to explain the main components of qtile layouts, how they work, and how you can use them to create your own layouts or hack existing layouts to make them work the way you want them.

Note: It is highly recommended that users wishing to create their own layout refer to the source documentation of existing layouts to familiarise themselves with the code.

20.1 What is a layout?

In Qtile, a layout is essentially a set of rules that determine how windows should be displayed on the screen. The layout is responsible for positioning all windows other than floating windows, "static" windows, internal windows (e.g. the bar) and windows that have requested not to be managed by the window manager.

20.2 Base classes

To simplify the creation of layouts, a couple of base classes are available to users.

20.2.1 The `Layout` class

As a bare minimum, all layouts should inherit the base `Layout` class object as this class defines a number of methods required for basic usage and will also raise errors if the required methods are not implemented. Further information on these required methods is set out below.

20.2.2 The `_SimpleLayoutBase` class

This class implements everything needed for a basic layout with the exception of the `configure` method. Therefore, unless your layout requires special logic for updating and navigating the list of clients, it is strongly recommended that your layout inherits this base class

20.2.3 The _ClientList class

This class defines a list of clients and the current client.

The collection is meant as a base or utility class for special layouts, which need to maintain one or several collections of windows, for example Columns or Stack, which use this class as base for their internal helper.

The property `current_index` get and set the index to the current client, whereas `current_client` property can be used with clients directly.

20.3 Required methods

To create a minimal, functioning layout your layout must include the methods listed below:

- `__init__`
- `configure`
- `add_client`
- `remove`
- `focus_first`
- `focus_last`
- `focus_next`
- `focus_previous`
- `next`
- `previous`

As noted above, if you create a layout based on the `_SimpleLayoutBase` class, you will only need to define `configure` (and `_init__`, if you have custom parameters). However, for the purposes of this document, we will show examples of all required methods.

20.3.1 __init__

Initialise your layout's variables here. The main use of this method will be to load any default parameters defined by layout. These are defined in a class attribute called `defaults`. The format of this attribute is a list of tuples.

```
from libqtile.layout import base


class TwoByTwo(base.Layout):
    """
    A simple layout with a fixed two by two grid.

    By default, unfocused windows are smaller than the focused window.
    """
    defaults = [
        ("border_width", 5, "Window border width"),
        ("border_colour", "00ff00", "Window border colour"),
        ("margin_focused", 5, "Margin width for focused windows"),
        ("margin_unfocused", 50, "Margin width for unfocused windows")
```

(continues on next page)

(continued from previous page)

```
    ]

    def __init__(self, **config):
        base.Layout.__init__(self, **config)
        self.add_defaults(TwoByTwo.defaults)
        self.clients = []
        self.current_client = None
```

Once the layouts is initialised, these parameters are available at `self.border_width` etc.

20.3.2 configure

This is where the magic happens! This method is responsible for determining how to position a window on the screen.

This method should therefore configure the dimensions and borders of a window using the window's `.place()` method. The layout can also call either `hide()` or `.unhide()` on the window.

```
def configure(self, client: Window, screen_rect: ScreenRect) -> None:
    """Simple example breaking screen into four quarters."""
    try:
        index = self.clients.index(client)
    except ValueError:
        # Layout not expecting this window so ignore it
        return

    # We're only showing first 4 windows
    if index > 3:
        client.hide()
        return

    # Unhide the window in case it was hiddent before
    client.unhide()

    # List to help us calculate x and y values of
    quarters = [
        (0, 0),
        (0.5, 0),
        (0, 0.5),
        (0.5, 0.5)
    ]

    # Calculate size and position for each window
    xpos, ypos = quarters[index]

    x = int(screen_rect.width * xpos) + screen_rect.x
    y = int(screen_rect.height * ypos) + screen_rect.y
    w = screen_rect.width // 2
    h = screen_rect.height // 2

    if client is self.current_client:
        margin = self.margin_focused
    else:
```

(continues on next page)

(continued from previous page)

```
        margin = self.margin_unfocused

    client.place(
        x,
        y,
        w - self.border_width * 2,
        h - self.border_width * 2,
        self.border_width,
        self.border_colour,
        margin=[margin] * 4,
    )
```

20.3.3 add_client

This method is called whenever a window is added to the group, regardless of whether the layout is current or not. The layout should just add the window to its internal datastructures, without mapping or configuring/displaying.

```
def add_client(self, client: Window) -> None:
    # Assumes self.clients is simple list
    self.clients.insert(0, client)
    self.current_client = client
```

20.3.4 remove

This method is called whenever a window is removed from the group, regardless of whether the layout is current or not. The layout should just de-register the window from its data structures, without unmapping the window.

The method must also return the "next" window that should gain focus or `None` if there are no other windows.

```
def remove(self, client: Window) -> Window | None:
    # Assumes self.clients is a simple list
    # Client already removed so ignore this
    if client not in self.clients:
        return None
    # Client is only window in the list
    elif len(self.clients) == 1:
        self.clients.remove(client)
        self.current_client = None
        # There are no other windows so return None
        return None
    else:
        # Find position of client in our list
        index = self.clients.index(client)
        # Remove client
        self.clients.remove(client)
        # Ensure the index value is not greater than list size
        # i.e. if we closed the last window in the list, we need to return
        # the first one (index 0).
        index %= len(self.clients)
        next_client = self.clients[index]
```

(continues on next page)

(continued from previous page)

```
        self.current_client = next_client
        return next_client
```

20.3.5 focus_first

This method is called when the first client in the layout should be focused.

This method should just return the first client in the layout, if any. NB the method should not focus the client itself, this is done by caller.

```
def focus_first(self) -> Window | None:
    if not self.clients:
        return None

    return self.client[0]
```

20.3.6 focus_last

This method is called when the last client in the layout should be focused.

This method should just return the last client in the layout, if any. NB the method should not focus the client itself, this is done by caller.

```
def focus_last(self) -> Window | None:
    if not self.clients:
        return None

    return self.client[-1]
```

20.3.7 focus_next

This method is called the next client in the layout should be focused.

This method should return the next client in the layout, if any. NB the layout should not cycle clients when reaching the end of the list as there are other method in the group for cycling windows which focus floating windows once the the end of the tiled client list is reached.

In addition, the method should not focus the client.

```
def focus_next(self, win: Window) -> Window | None:
    try:
        return self.clients[self.clients.index(win) + 1]
    except IndexError:
        return None
```

20.3.8 focus_previous

This method is called the previous client in the layout should be focused.

This method should return the previous client in the layout, if any. NB the layout should not cycle clients when reaching the end of the list as there are other method in the group for cycling windows which focus floating windows once the the end of the tiled client list is reached.

In addition, the method should not focus the client.

```
def focus_previous(self, win: Window) -> Window | None:
    if not self.clients or self.clients.index(win) == 0
        return None

    try:
        return self.clients[self.clients.index(win) - 1]
    except IndexError:
        return None
```

20.3.9 next

This method focuses the next tiled window and can cycle back to the beginning of the list.

```
def next(self) -> None:
    if self.current_client is None:
        return
    # Get the next client or, if at the end of the list, get the first
    client = self.focus_next(self.current_client) or self.focus_first()
    self.group.focus(client, True)
```

20.3.10 previous

This method focuses the previous tiled window and can cycle back to the end of the list.

```
def previous(self) -> None:
    if self.current_client is None:
        return
    # Get the previous client or, if at the end of the list, get the last
    client = self.focus_previous(self.current_client) or self.focus_last()
    self.group.focus(client, True)
```

20.4 Additional methods

While not essential to implement, the following methods can also be defined:

- clone
- show
- hide
- swap

- focus
- blur

20.4.1 clone

Each group gets a copy of the layout. The `clone` method is used to create this copy. The default implementation in `Layout` is as follows:

```
def clone(self, group: _Group) -> Self:
    c = copy.copy(self)
    c._group = group
    return c
```

20.4.2 show

This method can be used to run code when the layout is being displayed. The method receives one argument, the `ScreenRect` for the screen showing the layout.

The default implementation is a no-op:

```
def show(self, screen_rect: ScreenRect) -> None:
    pass
```

20.4.3 hide

This method can be used to run code when the layout is being hidden.

The default implementation is a no-op:

```
def hide(self) -> None:
    pass
```

20.4.4 swap

This method is used to change the position of two windows in the layout.

```
def swap(self, c1: Window, c2: Window) -> None:
    if c1 not in self.clients and c2 not in self.clients:
        return

    index1 = self.clients.index(c1)
    index2 = self.clients.index(c2)

    self.clients[index1], self.clients[index2] = self.clients[index2], self.
↪clients[index1]
```

20.4.5 focus

This method is called when a given window is being focused.

```
def focus(self, client: Window) -> None:
    if client not in self.clients:
        self.current_client = None
        return

    index = self.clients.index(client)

    # Check if window is not visible
    if index > 3:
        c = self.clients.pop(index)
        self.clients.insert(0, c)

    self.current_client = client
```

20.4.6 blur

This method is called when the layout loses focus.

```
def blur(self) -> None:
    self.current_client = None
```

20.5 Adding commands

Adding commands allows users to modify the behaviour of the layout. To make commands available via the command interface (e.g. via `lazy.layout` calls), the layout must include the following import:

```
from libqtile.command.base import expose_command
```

Commands are then decorated with `@expose_command`. For example:

```
@expose_command
def rotate(self, clockwise: bool = True) -> None:
    if not self.clients:
        return

    if clockwise:
        client = self.clients.pop(-1)
        self.clients.insert(0, client)
    else:
        client = self.clients.pop(0)
        self.clients.append(client)

    # Check if current client has been rotated off the screen
    if self.current_client and self.clients.index(self.current_client) > 3:
        if clockwise:
            self.current_client = self.clients[3]
```

(continues on next page)

(continued from previous page)

```
        else:
            self.current_client = self.clients[0]

    # Redraw the layout
    self.group.layout_all()
```

20.5.1 The `info` command

Layouts should also implement an `info` method to provide information about the layout.

As a minimum, the test suite (see below) will expect a layout to return the following information:

- Its name
- Its group
- The clients managed by the layout

NB the last item is not included in `Layout`'s implementation of the method so it should be added when defining a class that inherits that base.

```
@expose_command
def info(self) -> dict[str, Any]:
    inf = base.Layout.info(self)
    inf["clients"] = self.clients
    return inf
```

20.6 Adding layout to main repo

If you think your layout is amazing and you want to share with other users by including it in the main repo then there are a couple of extra steps that you need to take.

20.6.1 Add to list of layouts

You must save the layout in `libqtile/layout` and then add a line importing the layout definition to `libqtile/layout/__init__.py` e.g.

```
from libqtile.layout.twobytwo import TwoByTwo
```

20.6.2 Add tests

Basic functionality for all layouts is handled automatically by the core test suite. However, you should create tests for any custom functionality of your layout (e.g. testing the `rotate` command defined above).

20.7 Full example

The full code for the example layout is as follows:

```
from __future__ import annotations

from typing import TYPE_CHECKING

from libqtile.command.base import expose_command
from libqtile.layout import base

if TYPE_CHECKING:
    from libqtile.backend.base import Window
    from libqtile.config import ScreenRect
    from libqtile.group import _Group


class TwoByTwo(base.Layout):
    """
    A simple layout with a fixed two by two grid.

    By default, unfocused windows are smaller than the focused window.
    """
    defaults = [
        ("border_width", 5, "Window border width"),
        ("border_colour", "00ff00", "Window border colour"),
        ("margin_focused", 5, "Margin width for focused windows"),
        ("margin_unfocused", 50, "Margin width for unfocused windows")
    ]

    def __init__(self, **config):
        base.Layout.__init__(self, **config)
        self.add_defaults(TwoByTwo.defaults)
        self.clients = []
        self.current_client = None

    def configure(self, client: Window, screen_rect: ScreenRect) -> None:
        """Simple example breaking screen into four quarters."""
        try:
            index = self.clients.index(client)
        except ValueError:
            # Layout not expecting this window so ignore it
            return

        # We're only showing first 4 windows
        if index > 3:
            client.hide()
            return

        # Unhide the window in case it was hiddent before
        client.unhide()

        # List to help us calculate x and y values of
```

(continues on next page)

(continued from previous page)

```
        quarters = [
            (0, 0),
            (0.5, 0),
            (0, 0.5),
            (0.5, 0.5)
        ]

        # Calculate size and position for each window
        xpos, ypos = quarters[index]

        x = int(screen_rect.width * xpos) + screen_rect.x
        y = int(screen_rect.height * ypos) + screen_rect.y
        w = screen_rect.width // 2
        h = screen_rect.height // 2

        if client is self.current_client:
            margin = self.margin_focused
        else:
            margin = self.margin_unfocused

        client.place(
            x,
            y,
            w - self.border_width * 2,
            h - self.border_width * 2,
            self.border_width,
            self.border_colour,
            margin=[margin] * 4,
        )

    def add_client(self, client: Window) -> None:
        # Assumes self.clients is simple list
        self.clients.insert(0, client)
        self.current_client = client

    def remove(self, client: Window) -> Window | None:
        # Assumes self.clients is a simple list
        # Client already removed so ignore this
        if client not in self.clients:
            return None
        # Client is only window in the list
        elif len(self.clients) == 1:
            self.clients.remove(client)
            self.current_client = None
            # There are no other windows so return None
            return None
        else:
            # Find position of client in our list
            index = self.clients.index(client)
            # Remove client
            self.clients.remove(client)
            # Ensure the index value is not greater than list size
```

(continues on next page)

(continued from previous page)

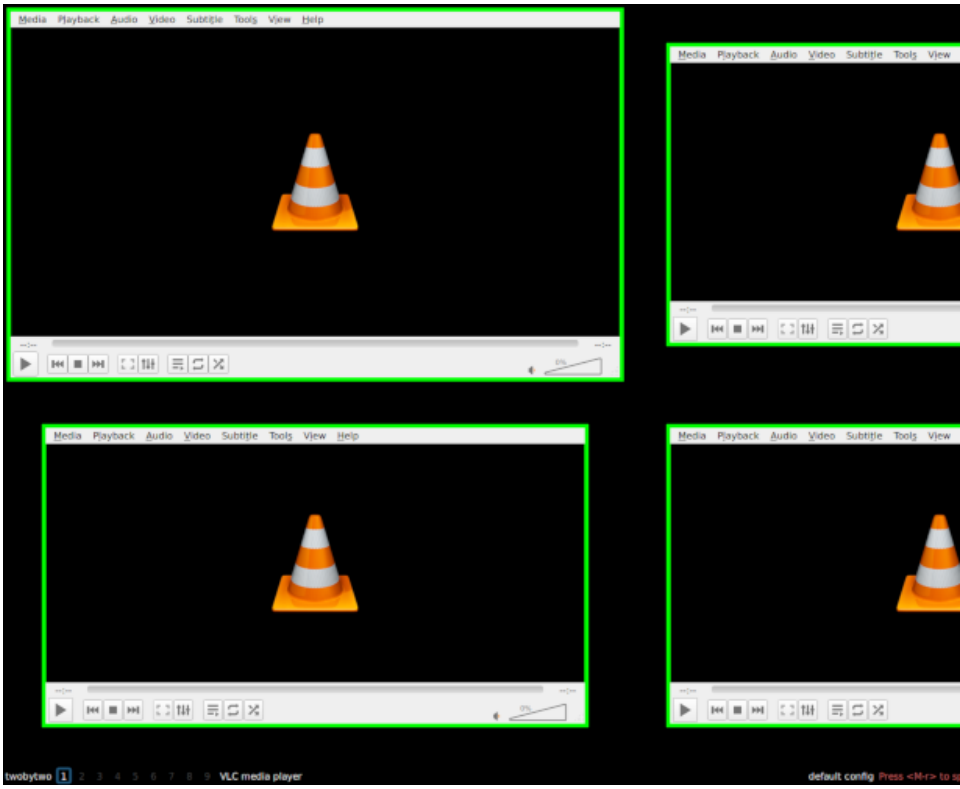
```
            # i.e. if we closed the last window in the list, we need to return
            # the first one (index 0).
            index %= len(self.clients)
            next_client = self.clients[index]
            self.current_client = next_client
            return next_client

    def focus_first(self) -> Window | None:
        if not self.clients:
            return None

        return self.client[0]

    def focus_last(self) -> Window | None:
        if not self.clients:
            return None

        return self.client[-1]

    def focus_next(self, win: Window) -> Window | None:
        try:
            return self.clients[self.clients.index(win) + 1]
        except IndexError:
            return None

    def focus_previous(self, win: Window) -> Window | None:
        if not self.clients or self.clients.index(win) == 0:
            return None

        try:
            return self.clients[self.clients.index(win) - 1]
        except IndexError:
            return None

    def next(self) -> None:
        if self.current_client is None:
            return
        # Get the next client or, if at the end of the list, get the first
        client = self.focus_next(self.current_client) or self.focus_first()
        self.group.focus(client, True)

    def previous(self) -> None:
        if self.current_client is None:
            return
        # Get the previous client or, if at the end of the list, get the last
        client = self.focus_previous(self.current_client) or self.focus_last()
        self.group.focus(client, True)

    def swap(self, c1: Window, c2: Window) -> None:
        if c1 not in self.clients and c2 not in self.clients:
            return
```

(continues on next page)

(continued from previous page)

```
        index1 = self.clients.index(c1)
        index2 = self.clients.index(c2)

        self.clients[index1], self.clients[index2] = self.clients[index2], self.
↪clients[index1]

    def focus(self, client: Window) -> None:
        if client not in self.clients:
            self.current_client = None
            return

        index = self.clients.index(client)

        # Check if window is not visible
        if index > 3:
            c = self.clients.pop(index)
            self.clients.insert(0, c)

        self.current_client = client

    def blur(self) -> None:
        self.current_client = None

    @expose_command
    def rotate(self, clockwise: bool = True) -> None:
        if not self.clients:
            return

        if clockwise:
            client = self.clients.pop(-1)
            self.clients.insert(0, client)
        else:
            client = self.clients.pop(0)
            self.clients.append(client)

        # Check if current client has been rotated off the screen
        if self.current_client and self.clients.index(self.current_client) > 3:
            if clockwise:
                self.current_client = self.clients[3]
            else:
                self.current_client = self.clients[0]

        # Redraw the layout
        self.group.layout_all()

    @expose_command
    def info(self) -> dict[str, Any]:
        inf = base.Layout.info(self)
        inf["clients"] = self.clients
        return inf
```

This should result in a layout looking like this:



20.8 Getting help

If you still need help with developing your widget then please submit a question in the qtile-dev group or submit an issue on the github page if you believe there's an error in the codebase.

CHAPTER

TWENTYONE

USING GIT

`git` is the version control system that is used to manage all of the source code. It is very powerful, but might be frightening at first. This page should give you a quick overview, but for a complete guide you will have to search the web on your own. Another great resource to get started practically without having to try out the newly-learned commands on a pre-existing repository is learn git branching. You should probably learn the basic `git` vocabulary and then come back to find out how you can use all that practically. This guide will be oriented on how to create a pull request and things might be in a different order compared to the introductory guides.

Warning: This guide is not complete and never will be. If something isn't clear, consult other sources until you are confident you know what you are doing.

21.1 I want to try out a feature somebody is working on

If you see a pull request on GitHub that you want to try out, have a look at the line where it says:

```
user wants to merge n commits into qtile:master from user:branch
```

Right now you probably have one *remote* from which you can fetch changes, the `origin`. If you cloned `qtile/qtile`, `git remote show origin` will spit out the *upstream* url. If you cloned your fork, `origin` points to it and you probably want to `git remote add upstream https://www.github.com/qtile/qtile`. To try out somebody's work, you can add their fork as a new remote:

```
git remote add <user> https://www.github.com/user/qtile
```

where you fill in the username from the line we asked you to search for before. Then you can load data from that remote with `git fetch` and then ultimately check out the branch with `git checkout <user>/<branch>`.

Alternatively, it is also possible to fetch and checkout pull requests without needing to add other remotes. The upstream remote is sufficient:

```
git fetch upstream pull/<id>/head:pr<id>
git checkout pr<id>
```

The numeric pull request id can be found in the url or next to the title (preceeded by a # symbol).

Note: Having the feature branch checked out doesn't mean that it is installed and will be loaded when you restart qtile. You might still need to install it with `pip`.

21.2 I committed changes and the tests failed

You can easily change your last commit: After you have done your work, `git add` everything you need and use `git commit --amend` to change your last commit. This causes the git history of your local clone to be diverged from your fork on GitHub, so you need to force-push your changes with:

```
git push -f <origin> <feature-branch>
```

where origin might be your user name or `origin` if you cloned your fork and feature-branch is to be replaced by the name of the branch you are working on.

Assuming the feature branch is currently checked out, you can usually omit it and just specify the origin.

21.3 I was told to rebase my work

If *upstream/master* is changed and you happened to change the same files as the commits that were added upstream, you should rebase your work onto the most recent *upstream/master*. Checkout your master, pull from *upstream*, checkout your branch again and then rebase it:

```
git checkout master
git pull upstream/master
git checkout <feature-branch>
git rebase upstream/master
```

You will be asked to solve conflicts where your diff cannot be applied with confidence to the work that was pushed upstream. If that is the case, open the files in your text editor and resolve the conflicts manually. You possibly need to `git rebase --continue` after you have resolved conflicts for one commit if you are rebasing multiple commits.

Note that the above doesn't work if you didn't create a branch. In that case you will find guides elsewhere to fix this problem, ideally by creating a branch and resetting your master branch to where it should be.

21.4 I was told to squash some commits

If you introduce changes in one commit and replace them in another, you are told to squash these changes into one single commit without the intermediate step:

```
git rebase -i master
```

opens a text editor with your commits and a comment block reminding you what you can do with your commits. You can reword them to change the commit message, reorder them or choose `fixup` to squash the changes of a commit into the commit on the line above.

This also changes your git history and you will need to force-push your changes afterwards.

Note that interactive rebasing also allows you to split, reorder and edit commits.

21.5 I was told to edit a commit message

If you need to edit the commit message of the last commit you did, use:

```
git commit --amend
```

to open an editor giving you the possibility to reword the message. If you want to reword the message of an older commit or multiple commits, use `git rebase -i` as above with the `reword` command in the editor.

CHAPTER

TWENTYTWO

LICENSE

This project is distributed under the MIT license.

Copyright (c) 2008, Aldo Cortesi All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

TWENTYTHREE

CHANGELOG

```
Qtile x.xx.x, released XXXX-XX-XX:
    !!! Python version support !!!
      - this release drops support for python 3.9
    * features
      - automatically lift types to their annotated type when specified via
        the `qtile cmd-obj` command line
    * bugfixes

Qtile 0.25.0, released 2024-04-06:
    * features
      - The Battery widget now supports dynamic charge control, allowing for
        protecting battery life.
      - To support the above (plus the other widgets that modify sysfs), qtile
        now ships with its own udev rules, located at
        /resources/99-qtile.rules; distro packagers will probably want to
        install this rule set.
    * bugfixes
      - Fix groups marked with `persist=False` not being deleted when their last window␣
↪is moved to another group.
      - Fallback icon in StatusNotifier widget

Qtile 0.24.0, released 2024-01-20:
    !!! config breakage/changes !!!
      - Matches no longer use "include/substring" style matching. But match the string␣
↪exactly. Previously on X11, if the WM_TYPE of a spawned window is e.g. dialog a match␣
↪with wm_type dialognoonereadschangelogs would return true. Additionally a window with␣
↪an empty WM_CLASS (which can happen) would match anything. If you rely this style of␣
↪substring matching, pass a regex to your match or use a function with func=.
        Using a list of strings inside Match with role, title, wm_class, wm_instance_
↪class, wm_type are also deprecated, use a regex. Right now we replace the property␣
↪with a regex if it's a list and warn with a deprecation message. You can use "qtile␣
↪migrate" to migrate your config to this.
    * features
      - Change how `tox` runs tests. See https://docs.qtile.org/en/latest/manual/
↪contributing.html#running-tests-locally
      for more information on how to run tests locally.
      - Add `ScreenSplit` layout which allows multiple layouts per screen. Also adds␣
↪`ScreenSplit`
        widget to display name of active split.
      - Updated `Bluetooth` widget which allows users to manage multiple devices in a␣
```

(continues on next page)

(continued from previous page)

```
↪single widget
      - Add `align` option to `Columns` layout so new windows can be added to left or␣
↪right column.
      - `.when()` have two new parameters:
        - `func: Callable`: Enable call when the result of the callable evaluates to True
        - `condition: bool`: a boolean value to determine whether the lazy object should␣
↪be run. Unlike `func`, the
          condition is evaluated once when the config file is first loaded.
      - Add ability to have bar drawns over windows by adding `reserve=False` to bar's␣
↪config to
        stop the bar reserving screen space.
      - Add ability for third-party code (widgets, layouts) to create hooks
      - Add ability to create user-defined hooks which can be fired from external scripts
    * bugfixes
      - Fix two bugs in stacking transient windows in X11
      - Checking configs containing `qtile.core.name` with `python config.py` don't fail␣
↪anymore (but `qtile.core.name`
        will be `None`)
      - Fix an error if a wayland xwindow has unknown wm_type

Qtile 0.23.0, released 2023-09-24:
    !!! Dependency Changes !!!
      - xcffib must be upgraded to >= 1.4.0
      - cairocffi must be upgraded to >= 1.6.0
      - New optional dependency `pulsectl-asyncio` required for `PulseVolume` widget
    !!! Notice for packagers - wlroots (optional dependency) bump !!!
      - Qtile's wayland backend now requires on wlroots 0.16 (and pywlroots 0.16)
    !!! config breakage/changes !!!
      - The `cmd_` prefix has been dropped from all commands (this means command names␣
↪are common when accessed
        via the command interface or internal python objects).
      - Custom widgets should now expose command methods with the `@expose_command`␣
↪decorator (available via
        `from libqtile.command.base import expose_command`).
      - Some commands have been renamed (in addition to dropping the 'cmd_' prefix):
          `hints` -> `get_hints`
          `groups` -> `get_groups`
          `screens` -> `get_screens`
      - Layouts need to rename some methods:
        - `add` to `add_client`
        - `cmd_next` to `next`
        - `cmd_previous` to `previous`
      - Layouts or widgets that redefine the `commands` property need to update the␣
↪signature:
          `@expose_command()`
          `def commands(self) -> list[str]:`
      - `Window.getsize` has been renamed `Window.get_size` (i.e. merged with the get_
↪size command).
      - `Window.getposition` has been renamed `Window.get_position` (i.e. merged with␣
↪the get_position command).
      - The `StockTicker` widget `function` option is being deprecated: rename it to␣
↪`func`.
```

(continues on next page)

(continued from previous page)

```
     - The formatting of `NetWidget` has changed, if you use the `format` parameter in␣
↪your config include
       `up_suffix`, `total_suffix` and `down_suffix` to display the respective units.
     - The `Notify` widget now has separate `default_timeout` properties for differenct␣
↪urgency levels. Previously,
       `default_timeout` was `None` which meant that there was no timeout for all␣
↪notifications (unless this had been
       set by the client sending the notification). Now, `default_timeout` is for␣
↪normal urgency notifications and this
       has been set to a default of 10 seconds. `default_timeout_urgent`, for critical␣
↪notifications, has a timeout of `None`.
     - The `PulseVolume` widget now depends on a third party library, `pulsectl-
↪asyncio`, to interact with the pulse audio
       server. Users will now see an `ImportError` until they install that library.
   * features
       - Add ability to set icon size in `LaunchBar` widget.
       - Add 'warp_pointer' option to `Drag` that when set will warp the pointer to the␣
↪bottom right of
         the window when dragging begins.
       - Add `currentsong` status to `Mpd2` widget.
       - Add ability to disable group toggling in `GroupBox` widget
       - Add ability to have different border color when windows are stacked in Stack␣
↪layout. Requires
         setting `border_focus_stack` and `border_normal_stack` variables.
       - Add ability to have different single border width for Columns layout by␣
↪setting 'single_border_width' key.
       - Add ability to have different border and margin widths when VerticalTile␣
↪layout only contains 1 window by
         setting 'single_border_width' and 'single_margin' keys.
       - New widget: GenPollCommand
       - Add `format` and `play_icon` parameters for styling cmus widget.
       - Add ability to add a group at a specified index
       - Add ability to spawn the `WidgetBox` widget opened.
       - Add ability to swap focused window based on index, and change the order of␣
↪windows inside current group
       - Add ability to update the widget only once if `update_interval` is None.
       - Add `move_to_slice` command to move current window to single layout in `Slice`␣
↪layout
       - Made the `NetWidget` text formattable.
       - Qtile no longer floods the log following X server disconnection, instead␣
↪handling those errors.
       - `Key` and `KeyChord` bindings now have another argument `swallow`.
         It indicates whether or not the pressed keys should be passed on to the␣
↪focused client.
         By default the keys are not passed (swallowed), so this argument is set to␣
↪`True`.
         When set to `False`, the keys are passed to the focused client. A key is never␣
↪swallowed if the
         function is not executed, e.g. due to failing the `.when()` check.
       - Add ability to set custom "Undefined" status key value to `Mpd2Widget`.
       - `Mpd2Widget` now searches for artist name in all similar keys (i.e␣
↪`albumartist`, `performer`, etc.).
```

(continues on next page)

(continued from previous page)

```
        - Add svg support to `CustomLayoutIcon`
        - added layering controls for X11 (Wayland support coming soon!):
          - `lazy.window.keep_above()/keep_below()` marks windows to be kept above/below␣
↪other windows permanently.
             Calling the functions with no arguments toggles the state, otherwise pass␣
↪`enable=True` or `enable=False`.
          - `lazy.window.move_up()/move_down()` moves windows up and down the z axis.
          - added `only_focused` setting to Max layout, allowing to draw multiple␣
↪clients on top of each other when
            set to False
        - Add `suspend` hook to run functions before system goes to sleep.
    * bugfixes
        - Fix bug where Window.center() centers window on the wrong screen when using␣
↪multiple monitors.
        - Fix `Notify` bug when apps close notifications.
        - Fix `CPU` precision bug with specific version of `psutil`
        - Fix config being reevaluated twice during reload (e.g. all hooks from config␣
↪were doubled)
        - Fix `PulseVolume` high CPU usage when update_interval set to 0.
        - Fix `Battery` widget on FreeBSD without explicit `battery` index given.
        - Fix XMonad layout faulty call to nonexistent _shrink_up
        - Fix setting tiled position by mouse for layouts using _SimpleLayoutBase. To␣
↪support this in other layouts, add a swap method taking two windows.
        - Fix unfullscreening bug in conjunction with Chromium based clients when auto_
↪fullscreen is set to `False`.
        - Ensure `CurrentLayoutIcon` expands paths for custom folders.
        - Fix vertical alignment of icons in `TaskList` widget
        - Fix laggy resize/positioning of floating windows in X11 by handling motion␣
↪notify events later. We also introduced a cap setting if you want to limit these␣
↪events further, e.g. for limiting resource usage. This is configurable with the x11_
↪drag_polling_rate variable for each `Screen` which is set to None by default,␣
↪indicating no cap.
    * python version support
        - We have added support for python 3.11 and pypy 3.9.
        - python 3.7, 3.8 and pypy 3.7 are not longer supported.
        - Fix bug where `StatusNotifier` does not update icons

Qtile 0.22.0, released 2022-09-22:
    !!! Config breakage !!!
        - lazy.qtile.display_kb() no longer receives any arguments. If you passed it any␣
↪arguments
          (which were ignored previously), remove them.
        - If you have a custom startup Python script that you use instead of `qtile␣
↪start` and run init_log
          manually, the signature has changed. Please check the source for the updated␣
↪arguments.
        - `KeyChord`'s signature has changed. ``mode`` is now a boolean to indicate␣
↪whether the mode should persist.
          The ``name`` parameter should be used to name the chord (e.g. for the␣
↪``Chord`` widget).
    * features
        - Add ability to draw borders and add margins to the `Max` layout.
```

(continues on next page)

(continued from previous page)

```
        - The default XWayland cursor is now set at startup to left_ptr, so an xsetroot␣
↪call is not needed to
          avoid the ugly X cursor.
        - Wayland: primary clipboard should now behave same way as with X after␣
↪selecting something it
          should be copied into clipboard
        - Add `resume` hook when computer resumes from sleep/suspend/hibernate.
        - Add `text_only` option for `LaunchBar` widget.
        - Add `force_update` command to `ThreadPoolText` widgets to simplify updating␣
↪from key bindings
        - Add scrolling ability to `_TextBox`-based widgets.
        - Add player controls (via mouse callbacks) to `Mpris2` widget.
        - Wayland: input inhibitor protocol support added (pywayland>=0.4.14 & pywlroots>
↪=0.15.19)
        - Add commands to control Pomodoro widget.
        - Add icon theme support to `TaskList` widget (available on X11 and Wayland␣
↪backends).
        - Wayland: Use `qtile cmd-obj -o core -f get_inputs` to get input device␣
↪identifiers for
          configuring inputs. Also input configs will be updated by config reloads␣
↪(pywlroots>=0.15.21)
    * bugfixes
        - Widgets that are incompatible with a backend (e.g. Systray on Wayland) will no␣
↪longer show
          as a ConfigError in the bar. Instead the widget is silently removed from the␣
↪bar and a message
          included in the logs.
        - Reduce error messages in `StatusNotifier` widget from certain apps.
        - Reset colours in `Chord` widget
        - Prevent crash in `LaunchBar` when using SVG icons
        - Improve scrolling in `Mpris2` widget (options to repeat scrolling etc.)

Qtile 0.21.0, released 2022-03-23:
    * features
        - Add `lazy.window.center()` command to center a floating window on the screen.
        - Wayland: added power-output-management-v1 protocol support, added idle␣
↪protocol,
          added idle inhibit protocol
        - Add MonadThreeCol layout based on XMonad's ThreeColumns.
        - Add `lazy.screen.set_wallpaper` command.
        - Added ability to scale the battery icon's size
        - Add Spiral layout
        - Add `toggle` argument to `Window.togroup` with the same functionality as in␣
↪`Group.toscreen`.
        - Added `margin_on_single` and `border_on_single` to Bsp layout
    * bugfixes
        - Fix `Systray` crash on `reconfigure_screens`.
        - Fix bug where widgets can't be mirrored in same bar.
        - Fix various issues with setting fullscreen windows floating and vice versa.
        - Fix a bug where a .when() check for lazy functions errors out when matching
          on focused windows when none is focused. By default we do not match on focused␣
↪windows,
```

(continues on next page)

(continued from previous page)

```
          to change this set `if_no_focused` to True.
        - Widget with duplicate names will be automatically renamed by appending numeric␣
↪suffixes
        - Fix resizing of wallpaper when screen scale changes (X11)
        - Two small bugfixes for `StatusNotifier` - better handling of Ayatana indicators
        - Fix bug where StatusNotifierItem crashes due to invalid object paths (e.g.␣
↪Zoom)

Qtile 0.20.0, released 2022-01-24:
    * features
        - Add `place_right` option in the TreeTab layout to place the tab panel on the␣
↪right side
        - X11: Add support for _NET_DESKTOP_VIEWPORT. E.g. can be used by rofi to map on␣
↪current output.
        - Wayland: Bump wlroots version. 0.15.x wlroots and 0.15.2+ pywlroots are␣
↪required.
        - Add XWayland support to the Wayland backend. XWayland will start up as needed,␣
↪if it is installed.
    * bugfixes
        - Remove non-commandable windows from IPC. Fixes bug where IPC would fail when␣
↪trying to get info
          on all windows but Systray has icons (which are non-commandable `_Window`s.)
        - Fix bug where bars were not reconfigured correctly when screen layout changes.
        - Fix a Wayland bug where layer-shell surface like dunst would freeze up and␣
↪stop updating.
        - Change timing of `screens_reconfigured` hook. Will now be called ONLY if `cmd_
↪reconfigure_screens`
          has been called and completed.
        - Fix order of icons in Systray widget when restarting/reloading config.
        - Fix rounding error in PulseVolume widget's reported volume.
        - Fix bug where Volume widget did not load images where `theme_path` had been␣
↪set in `widget_defaults`.
        - Remove ability to have multiple `Systray` widgets. Additional `Systray`␣
↪widgets will result in a
          ConfigError.
        - Release notification name from dbus when finalising `Notify` widget. This␣
↪allows other notification
          managers to request the name.
        - Fix bug where `Battery` widget did not retrieve `background` from `widget_
↪defaults`.
        - Fix bug where widgets in a `WidgetBox` are rendered on top of bar borders.

        - Add ability to swap focused window based on index, and change the order of␣
↪windows inside current group
Qtile 0.19.0, released 2021-12-22:
    * features
        - Add ability to draw borders to the Bar. Can customise size and colour per edge.
        - Add `StatusNotifier` widget implementing the `StatusNotifierItem`␣
↪specification.
          NB Widget does not provide context menus.
        - Add `total` bandwidth format value to the Net widget.
        - Scratchpad groups could be defined as single so that only one of the␣
```

(continues on next page)

(continued from previous page)

```
↪scratchpad in the group is visible
          at a given time.
        - All scratchpads in a Scratchpad group can be hidden with hide_all() function.
        - For saving states of scratchpads during restart, we use wids instead of pids.
        - Scratchpads can now be defined with an optional matcher to match with window␣
↪properties.
        - `Qtile.cmd_reload_config` is added for reloading the config without completely␣
↪restarting.
        - Window.cmd_togroup's argument `groupName` should be changed to
          `group_name`. For the time being a log warning is in place and a
          migration is added. In the future `groupName` will fail.
        - Add `min/max_ratio` to Tile layout and fix bug where windows can extend␣
↪offscreen.
        - Add ability for widget `mouse_callbacks` to take `lazy` calls (similar to␣
↪keybindings)
        - Add `aliases` to `lazy.spawncmd()` which takes a dictionary mapping convenient␣
↪aliases
          to full command lines.
        - Add a new 'prefix' option to the net widget to display speeds with a static␣
↪unit (e.g. MB).
        - `lazy.group.toscreen()` now does not toggle groups by default. To get this␣
↪behaviour back, use
          `lazy.group.toscreen(toggle=True)`
        - Tile layout has new `margin_on_single` and `border_on_single` option to specify
          whether to draw margin and border when there is only one window.
        - Thermal zone widget.
        - Allow TextBox-based widgets to display in vertical bars.
        - Added a focused attribute to `lazy.function.when` which can be used to Match␣
↪on focused windows.
        - Allow to update Image widget with update() function by giving a new path.
    * bugfixes
        - Windows are now properly re-ordered in the layouts when toggled on and off␣
↪fullscreen

Qtile 0.18.1, released 2021-09-16:
    * features
        - All layouts will accept a list of colors for border_* options with which
          they will draw multiple borders on the appropriate windows.

Qtile 0.18.0, released 2021-07-04:
    !!! Config breakage !!!
        - The `qtile` entry point doesn't run `qtile start` by default anymore
        - New optional dependency for dbus related features: dbus-next.
          Replaces previous reliance on dbus/Glib and allows qtile to use async
          dbus calls within asyncio's eventloop.
        - widget.BatteryIcon no longer has a fallback text mode; use
          widget.Battery instead
        - MonadX layout key new_at_current is deprecated, use new_client_position.
        - `libqtile.window` has been moved to `libqtile.backend.x11.window`; a migration␣
↪has been added for this.
    !!! deprecation warning !!!
        - 'main' config functions, deprecated in 0.16.1, will no longer be executed.
```

(continues on next page)

(continued from previous page)

```
    !!! Notice for packagers - new dependencies !!!
        - Tests now require the 'dbus-next' python module plus 'dbus-launch' and 'notify-
↪send' applications
    * features
        - added transparency in x11 and wayland backends
        - added measure_mem and measure_swap attributes to memory widget to allow user␣
↪to choose measurement units.
        - memory widget can now be displayed with decimal values
        - new "qtile migrate" command, which will attempt to upgrade previous
          configs to the current version in the case of qtile API breaks.
        - A new `reconfigure_screens` config setting. When `True` (default) it
          hooks `Qtile.reconfigure_screens` to the `screen_change` hook,
          reconfiguring qtile's screens in response to randr events. This
          removes the need to restart qtile when adding/removing external
          monitors.
        - improved key chord / sequence functionality. Leaving a chord with `mode`
          set brings you to a named mode you activated before, see #2264.
          A new command, `lazy.ungrab_all_chords`, was introduced to return to the root␣
↪bindings.
          The `enter_chord` hook is now always called with a string argument.
          The third argument to `KeyChord` was renamed from `submaping` to `submapping`␣
↪(typo fix).
        - added new argument for CheckUpdates widget: `custom_command_modify` which␣
↪allows user to modify the
          the line count of the output of `custom_command` with a lambda function (i.e.␣
↪`lambda x: x-3`).
          Argument defaults to `lambda x: x` and is overridden by `distro` argument's␣
↪internal lambda.
        - added new argument for the WindowName, WindowTabs and Tasklist widgets: `parse_
↪text` which allows users to
          define a function that takes a window name as an input, modify it in some way␣
↪(e.g. str.replace(), str.upper() or regex)
          and show that modification on screen.
        - A Wayland backend has been added which can be used by calling `qtile start -b␣
↪wayland` directly in your TTY.
          It requires the latest releases of wlroots, python-xkbcommon, pywayland and␣
↪pywlroots. It is expected to be
          unstable so please let us know if you find any bugs!
        - The 'focus` argument to `Click` and `Drag` objects in your config are no␣
↪longer necessary (and are ignored).

Qtile 0.17.0, released 2021-02-13:
    !!! Python version breakage !!!
        - Python 3.5 and 3.6 are no longer supported
    !!! Config breakage !!!
        - Pacman widget has been removed. Use CheckUpdates instead.
        - Mpris widget has been removed. Use Mpris2 instead.
        - property "masterWindows" of Tile layout renamed to master_length
        - Match objects now only allow one string argument for their wm
          name/class/etc. properties. to update your config, do e.g.
                  Group('www', spawn='firefox', layout='xmonad',
              -           matches=[Match(wm_class=['Firefox', 'google-chrome', 'Google-
```

(continues on next page)

(continued from previous page)

```
↪chrome'])]),
             +          matches=[Match(wm_class='Firefox'), Match(wm_class='google-
↪chrome'), Match(wm_class='Google-chrome')]),
        - properties wname, wmclass and role of Slice-layout replaced by Match-
          type property "match"
        - rules specified in `layout.Floating`'s `float_rules` are now evaluated with
          AND-semantics instead of OR-semantics, i.e. if you specify 2 different
          property rules, both have to match
        - check the new `float_rules` for `floating_layout` in the default config and
          extend your own rules appropriately: some non-configurable auto-floating rules
          were made explicit and added to the default config
        - using `dict`s for `layout.Floating`'s `float_rules` is now deprecated, please
          use `config.Match` objects instead
        - `no_reposition_match` in `layout.Floating` has been removed; use the list of
          `config.Match`-objects `no_reposition_rules` instead
        - Command line has been modernized to a single entry point, the `qtile`
          binary. Translations are below:
                qtile     -> qtile start
                qtile-cmd -> qtile cmd-obj
                qtile-run -> qtile run-cmd
                qtile-top -> qtile top
                qshell    -> qtile shell
          iqshell and dqtile-cmd are no longer distributed with the
          package, as they were either user or developer scripts. Both are
          still available in the qtile repo in /scripts.

          Running `qtile` without arguments will continue to work for the
          forseeable future, but will be eventually deprecated. qtile prints a
          warning when run in this configuration.
        - Qtile.cmd_focus_by_click is no longer an available command.
        - Qtile.cmd_get_info is no longer an available command.
        - libqtile.command_* has been deprecated, it has been moved to
          libqtile.command.*
        - libqtile.widget.base.ThreadedPollText has been removed; out of tree
          widgets can use ThreadPoolText in the same package instead.
        - the YahooWeather widget was removed since Yahoo retired their free
          tier of the weather API
        - Deprecated hook `window_name_change` got removed, use
          `client_name_updated` instead.
        - show_state attribute from WindowName widget has been removed. Use format␣
↪attribute instead.
                show_state = True  -> format = '{state}{name}'
                show_state = False -> format = '{name}'
        - mouse_callbacks no longer receives the qtile object as an argument
          (they receive no arguments); import it via `from libqtile import
          qtile` instead.
    * features
        - new WidgetBox widget
        - new restart and shutdown hooks
        - rules specified in `layout.Floating`'s `float_rules` are now evaluated with
          AND-semantics, allowing for more complex and specific rules
        - Python 3.9 support
```

(continues on next page)

(continued from previous page)

```
        - switch to Github Actions for CI
        - Columns layout has new `margin_on_single` option to specify margin
          size when there is only one window (default -1: use `margin` option).
        - new OpenWeather widget to replace YahooWeather
        - new format attribute for WindowName widget
        - new max_chars attribute for WindowName widget
        - libqtile now exports type information
        - add a new `qtile check` subcommand, which will check qtile configs
          for various things:
              - validates configs against the newly exported type information
                if mypy is present in the environment
              - validates that qtile can import the config file (e.g. that
                syntax is correct, ends in a .py extension, etc.)
              - validates Key and Mouse mod/keysym arguments are ok.
        - Columns layout now enables column swapping by using swap_column_left and swap_
↪column_right
    !!! warning !!!
        - When (re)starting, Qtile passes its state to the new process in a
          file now, where previously it passed state directly as a string. This
          fixes a bug where some character encodings (i.e. in group names) were
          getting messed up in the conversion to/from said string. This change
          will cause issues if you update Qtile then restart it, causing the
          running old version to pass state in the previous format to the new
          process which recognises the new.

Qtile 0.16.1, released 2020-08-11:
    !!! Config breakage !!!
        - Hooks 'addgroup', 'delgroup' and 'screen_change' will no longer
          receive the qtile object as an argument. It can be accessed directly
          at libqtile.qtile.
    !!! deprecation warning !!!
        - defining a main function in your config is deprecated. You should use
          @hook.subscribe.startup_complete instead. If you need access to the
          qtile object, import it from libqtile directly.
    * bugfixes
        - include tests in the release for distros to consume
        - don't resize 0th screen incorrectly on root ConfigureNotify
        - expose qtile object as libqtile.qtile (note that we still consider
          anything not prefixed with cmd_ to be a private API)
        - fix transparent borders
        - MonadTall, MonadWide, and TreeTab now work with Slice

Qtile 0.16.0, released 2020-07-20:
    !!! Config breakage !!!
        - Imports from libqtile.widget are now made through a function
          proxy to avoid the side effects of importing all widgets at
          once. If you subclass a widget in your config, import it from
          its own module.
          e.g. from libqtile.widget.pomodoro import Pomodoro
    * features
        - added `guess_terminal` in utils
        - added keybinding cheet sheet image generator
```

(continues on next page)

(continued from previous page)

```
        - custom keyboardlayout display
        - added native support for key chords
        - validate config before restart and refuse to restart with a bad
          config
        - added a bunch of type annotations to config objects (more to come)
    * bugfixes
        - major focus rework; Java-based IDEs such as PyCharm, NetBrains, etc.
          now focus correctly
        - fix a bug where spotify (or any window with focus-to=parent) was
          closed, nothing would be focused and no hotkeys would work
        - support windows unsetting the input hint
        - respects window's/user's location setting if present (WM_SIZE_HINTS)
        - fixed YahooWeather widget for new API
        - fix a bug where _NET_WM_DESKTOPS wasn't correctly updated when
          switching screens in some cases
        - fix a crash in the BSP layout
        - fix a stacktrace when unknown keysyms are encounted
        - make qtile --version output more sane
        - fix a rendering issue with special characters in window names
        - keyboard widget no longer re-sets the keyboard settings every second
        - fix qtile-top with the new IPC model
        - Image widget respects its background setting now
        - correctly re-draw non-focused screens on qtile restart
        - fix a crash when decoding images
        - fix the .when() constraint for lazy objects

Qtile 0.15.1, released 2020-04-14
    * bugfixes
        - fix qtile reload (it was crashing)

Qtile 0.15.0, released 2020-04-12:
    !!! Config breakage !!!
        - removed the mpd widget, which depended on python-mpd.
        - the Clock widget now requires pytz to handle timezones that are
          passed as string
        - libqtile.command.Client does not exist anymore and has been
          replaced by libqtile.command_client.CommandClient
    !!! deprecation warning !!!
        - libqtile.command.lazy is deprecated in favor of libqtile.lazy.lazy
    * features
        - Python 3.8 support
        - `wallpaper` and `wallpaper_mode` for screens
        - bars can now have margins
        - `lazy.toscreen` called twice will now toggle the groups
          (optional with the `toggle` parameter)
        - `lazy.window.togroup` now has `switch_group` parameter to follow
          the window to the group it is sent to
        - qtile now copies the default config if the config file does not exist
        - all widgets now use Pango markup by default
        - add an `fmt` option for all textbox widgets
        - new PulseVolume widget for controlling PulseAudio
        - new QuickExit widget, mainly for the default config
```

(continues on next page)

(continued from previous page)

```
        - new non-graph CPU widget
        - KeyboardLayout widget: new `options` parameter
        - CheckUpdates widget: support ArchLinux yay
        - GroupBox widget: new `block_highlight_text_color` parameter
        - Mpd2 widget: new `color_progress` parameter
        - Maildir widget can now display the inbox grand total
        - the Net widget can now use bits as unit
        - Spacer widget: new `background_color` parameter
        - More consistent resize behavior in Columns layout
        - various improvements of the default config
        - large documentation update and improvements (e.g. widget
          dependencies)
    * bugfixes
        - qtile binary: don't fail if we can't set the locale
        - don't print help if qtile-cmd function returns nothing
        - Monad layout: fix margins when flipped

Qtile 0.14.2, released 2019-06-19:
    * bugfixes
        - previous release still exhibited same issues with package data,
          really fix it this time

Qtile 0.14.1, released 2019-06-19:
    * bugfixes
        - properly include png files in the package data to install included
          icons

Qtile 0.14.0, released 2019-06-19:
    !!! Python version breakage !!!
        - Python 2 is no longer supported
        - Python 3.4 and older is no longer supported
    !!! Config breakage !!!
        - Many internal things were renamed from camel case to snake case. If
          your config uses main(), or any lazy.function() invocations that
          interact directly with the qtile object, you may need to forward port
          them. Also note that we do *not* consider the qtile object to be a
          stable api, so you will need to continue forward porting these things
          for future refactorings (for wayland, etc.). A better approach may be
          to add an upstream API for what you want to do ;)
        - Maildir's subFolder and maildirPath changed to maildir_path and
          sub_folder.
        - the graph widget requires the psutil library to be installed
    * features
        - add custom `change_command` to backlight widget
        - add CommandSet extension to list available commands
        - simplify battery monitoring widget interface and add freebsd
          compatible battery widget implementation
        - track last known mouse coordinates on the qtile manager
        - allow configuration of warping behavior in columns layout
    * bugfixes
        - with cursor warp enabled, the cursor is warped on screen change
        - fix stepping groups to skip the scratch pad group
```

(continues on next page)

(continued from previous page)

```
        - fix stack layout to properly shuffle
        - silence errors when unmapping windows

Qtile 0.13.0, released 2018-12-23:
    !!! deprecation warning !!!
        - wmii layout is deprecated in terms of columns layout, which has the
          same behavior with different defaults, see the wmii definition for
          more details
    * features
        - add svg handling for images
        - allow addgroup command to set the layout
        - add command to get current log level
        - allow groupbox to hide unused groups
        - add caps lock indicator widget
        - add custom_command to check_update widget
    * bugfixes
        - better shutdown handling
        - fix clientlist current client tracking
        - fix typo in up command on ratiotile layout
        - various fixes to check_update widget
        - fix 0 case for resize screen

Qtile 0.12.0, released 2018-07-20:
    !!! Config breakage !!!
        - Tile layout commands up/down/shuffle_up/shuffle_down changed to be
          more consistent with other layouts
        - move qcmd to qtile-cmd because of conflict with renameutils, move
          dqcmd to dqtile-cmd for symmetry
    * features
        - add `add_after_last` option to Tile layout to add windows to the end
          of the list.
        - add new formatting options to TaskList
        - allow Volume to open app on right click
    * bugfixes
        - fix floating of file transfer windows and java drop-downs
        - fix exception when calling `cmd_next` and `cmd_previous` on layout
          without windows
        - fix caps lock affected behaviour of key bindings
        - re-create cache dir if it is deleted while qtile is running
        - fix CheckUpdates widget color when no updates
        - handle cases where BAT_DIR does not exist
        - fix the wallpaper widget when using `wallpaper_command`
        - fix Tile layout order to not reverse on reset
        - fix calling `focus_previous/next` with no windows
        - fix floating bug is BSP layout

Qtile 0.11.1, released 2018-03-01:
    * bug fix
        - fixed pip install of qtile

Qtile 0.11.0, released 2018-02-28:
    !!! Completely changed extension configuration, see the documentation !!!
```

(continues on next page)

(continued from previous page)

```
    !!! `extention` subpackage renamed to `extension` !!!
    !!! `extentions` configuration variable changed to `extension_defaults` !!!
    * features
        - qshell improvements
        - new MonadWide layout
        - new Bsp layout
        - new pomodoro widget
        - new stock ticker widget
        - new `client_name_updated` hook
        - new RunCommand and J4DmenuDesktop extension
        - task list expands to fill space, configurable via `spacing` parameter
        - add group.focus_by_name() and group.info_by_name()
        - add disk usage ratio to df widget
        - allow displayed group name to differ from group name
        - enable custom TaskList icon size
        - add qcmd and dqcmd to extend functionality around qtile.command
          functionality
        - add ScratchPad group that has configurable drop downs
    * bugfixes
        - fix race condition in Window.fullscreen
        - fix for string formatting in qtile_top
        - fix unicode literal in tasklist
        - move mpris2 initialization out of constructor
        - fix wlan widget variable naming and division
        - normalize behavior of layouts on various commands
        - add better fallback to default config
        - update btc widget to use coinbase
        - fix cursor warp when using default layout implementation
        - don't crash when using widget with unmet dependencies
        - fix floating window default location

Qtile 0.10.7, released 2017-02-14:
    * features
        - new MPD widget, widget.MPD2, based on `mpd2` library
        - add option to ignore duplicates in prompt widget
        - add additional margin options to GroupBox widget
        - add option to ignore mouse wheel to GroupBox widget
        - add `watts` formatting string option to Battery widgets
        - add volume commands to Volume widget
        - add Window.focus command
    * bugfixes
        - place transient windows in the middle of their parents
        - fix TreeTab layout
        - fix CurrentLayoutIcon in Python 3
        - fix xcb handling for xcffib 0.5.0
        - fix bug in Screen.resize
        - fix Qtile.display_kb command

Qtile 0.10.6, released 2016-05-24:
    !!! qsh renamed to qshell !!!
        This avoids name collision with other packages
    * features
```

(continues on next page)

(continued from previous page)

```
        - Test framework changed to pytest
        - Add `startup_complete` hook
    * bugfixes
        - Restore dynamic groups on restart
        - Correct placement of transient_for windows
        - Major bug fixes with floating window handling
    * file path changes (XDG Base Directory specification)
        - the default log file path changed from ~/.qtile.log to
          ~/.local/share/qtile/qtile.log
        - the cache directory changed from ~/.cache to ~/.cache/qtile
        - the prompt widget's history file changed from ~/.qtile_history to
          ~/.cache/qtile/prompt_history

Qtile 0.10.5, released 2016-03-06:
    !!! Python 3.2 support dropped !!!
    !!! GoogleCalendar widget dropped for KhalCalendar widget !!!
    !!! qtile-session script removed in favor of qtile script !!!
    * features
        - new Columns layout, composed of dynamic and configurable columns of
          windows
        - new iPython kernel for qsh, called iqsh, see docs for installing
        - new qsh command `display_kb` to show current key binding
        - add json interface to IPC server
        - add commands for resizing MonadTall main panel
        - wlan widget shows when you are disconnected and uses a configurable
          format
    * bugfixes
        - fix path handling in PromptWidget
        - fix KeyboardLayout widget cycling keyboard
        - properly guard against setting screen to too large screen index

Qtile 0.10.4, released 2016-01-19:
    !!! Config breakage !!!
        - positional arguments to Slice layout removed, now `side` and `width`
          must be passed in as keyword arguments
    * features
        - add alt coin support to BitcoinTracker widget
    * bugfixes
        - don't use six.moves assignment (fix for >=setuptools-19.3)
        - improved floating and fullscreen handling
        - support empty or non-charging secondary battery in BatteryWidget
        - fix GoogleCalendar widget crash

Qtile 0.10.3, released 2015-12-25:
    * features
        - add wmii layout
        - add BSD support to graph widgets
    * bugfixes
        - fix (some) fullscreen problems
        - update google calendar widget to latest google api
        - improve multiple keyboard layout support
        - fix displaying Systray widget on secondary monitor
```

(continues on next page)

(continued from previous page)

```
        - fix spawn file descriptor handling in Python 3
        - remove duplicate assert code in test_verticaltile.py
        - allow padding_{x,y} and margin_{x,y} widget attrs to be set to 0

Qtile 0.10.2, released 2015-10-19:
    * features
        - add qtile-top memory monitoring
        - GroupBox can set visible groups
        - new GroupBox highlighting, line
        - allow window state to be hidden on WindowName widget
        - cmd_togroup can move to current group when None sent
        - added MOC playback widget
        - added memory usage widget
        - log truncation, max log size, and number of log backups configurable
        - add a command to change to specific layout index
          (lazy.to_layout_index(index))
    * bugfixes
        - fixed memory leak in dgroups
        - margin fixes for MonalTall layout
        - improved cursor warp
        - remove deprecated imp for Python >= 3.3
        - properly close file for NetGraph
        - fix MondadTall layout grow/shrink secondary panes for Python 2
        - Clock widget uses datetime.now() rather than .fromtimestamp()
        - fix Python 3 compatibility of ThermalSensor widget
        - various Systray fixes, including implementing XEMBED protocol
        - print exception to log during loading config
        - fixed xmonad layout margins between main and secondary panes
        - clear last window name from group widgets when closed
        - add toggleable window border to single xmonad layout
    * config breakage
        - layouts.VerticalTile `windows` is now `clients`
        - layouts.VerticalTile focus_next/focus_previous now take a single
          argument, similar to other layouts

Qtile 0.10.1, released 2015-07-08:
    This release fixes a problem that made the PyPI package uninstallable,
    qtile will work with a pip install now

Qtile 0.10.0, released 2015-07-07:
    !!! Config breakage !!!
        - various deprecated commands have been removed:
            Screen.cmd_nextgroup: use cmd_next_group
            Screen.cmd_prevgroup: use cmd_prev_group
            Qtile.cmd_nextlayout: use cmd_next_layout
            Qtile.cmd_prevlayout: use cmd_prev_layout
            Qtile.cmd_to_next_screen: use cmd_next_screen
            Qtile.cmd_to_prev_screen: use cmd_prev_screen
        - Clock widget: remove fmt kwarg, use format kwarg
        - GmailChecker widget: remove settings parameter
        - Maildir widget: remove maildirPath, subFolders, and separator kwargs
    * Dependency updates
```

(continues on next page)

(continued from previous page)

```
        - cffi>=1.1 is now required, along with xcffib>=0.3 and cairocffi>=0.7
          (the cffi 1.0 compatible versions of each)
        - Care must be taken that xcffib is installed *before* cairocffi
    * features
        - add support for themed cursors using xcb-cursor if available
        - add CheckUpdate widget, for checking package updates, this deprecates
          the Pacman widget
        - add KeyboardKbdd widget, for changing keyboard layouts
        - add Cmus widget, for showing song playing in cmus
        - add Wallpaper widget, for showing and cycling wallpaper
        - add EzConfig classes allowing shortcuts to define key bindings
        - allow GroupBox urgent highlighting through text
        - Bar can be placed vertically on sides of screens (widgets must be
          adapted for vertical viewing)
        - add recognizing brightness keys
    * bugfixes
        - deprecation warnings were not printing to logs, this has been fixed
        - fix calculation of y property of Gap
        - fix focus after closing floating windows and floating windows
        - fix various Python 3 related int/float problems
        - remember screen focus across restarts
        - handle length 1 list passed to Drawer.set_source_rgb without raising
          divide by zero error
        - properly close files opened in Graph widget
        - handle _NET_WM_STATE_DEMANDS_ATTENTION as setting urgency
        - fix get_wm_transient_for, request WINDOW, not ATOM

Qtile 0.9.1, released 2015-02-13:
    This is primarily a unicode bugfix release for 0.9.0; there were several
    nits related to the python2/3 unicode conversion that were simply wrong.
    This release also adds license headers to each file, which is necessary for
    distro maintainers to package Qtile.
    * bugfixes
        - fix python2's importing of gobject
        - fix unicode handling in several places

Qtile 0.9.0, released 2015-01-20:
    * !!! Dependency Changes !!!
      New dependencies will need to be installed for Qtile to work
        - drop xpyb for xcffib (XCB bindings)
        - drop py2cairo for cairocffi (Cairo bindings)
        - drop PyGTK for asyncio (event loop, pangocairo bindings managed
          internally)
        - Qtile still depends on gobject if you want to use anything that uses
          dbus (e.g. the mpris widgets or the libnotify widget)
    * features
        - add Python 3 and pypy support (made possible by dependency changes)
        - new layout for vertical monitors
        - add startup_once hook, which is called exactly once per session (i.e.
          it is not called when qtile is restarted via lazy.restart()). This
          eliminates the need for the execute_once() function found in lots of
          user configs.
```

(continues on next page)

(continued from previous page)

```
        - add a command for showing/hiding the bar (lazy.hide_show_bar())
        - warn when a widget's dependencies cannot be imported
        - make qtile.log more useful via better warnings in general, including
          deprecation and various other warnings that were previously
          nonexistent
        - new text-polling widget super classes, which enable easy
          implementation of various widgets that need to poll things outside
          the event loop.
        - add man pages
        - large documentation update, widget/layout documentation is now
          autogenerated from the docstrings
        - new ImapWidget for checking imap mailboxes
    * bugfixes
        - change default wmname to "LG3D" (this prevents some java apps from
          not working out of the box)
        - all code passes flake8
        - default log level is now WARNING
        - all widgets now use our config framework
        - windows with the "About" role float by default
        - got rid of a bunch of unnecessary bare except: clauses

Qtile 0.8.0, released 2014-08-18:
    * features
        - massive widget/layout documentation update
        - new widget debuginfo for use in Qtile development
        - stack has new autosplit, fair options
        - matrix, ratiotile, stack, xmonad, zoomy get 'margin' option
        - new launchbar widget
        - support for matching WM_CLASS and pid in Match
        - add support for adding dgroups rules dynamically and via ipc
        - Clock supports non-system timezones
        - new mpris2 widget
        - volume widget can use emoji instead of numbers
        - add an 'eval' function to qsh at every object level
        - bar gradients support more colors
        - new Clipboard widget (very handy!)
    * bugfixes
        - bitcoin ticker widget switched from MtGox (dead) to btc-e
        - all widgets now use Qtile's defaults system, so their defaults are
          settable globally, etc.
        - fix behavior when screens are cloned
        - all widgets use a unified polling framework
        - "dialog" WM_TYPEs float by default
        - respect xrandr --primary
        - use a consistent font size in the default config
        - default config supports mouse movements and floating
        - fix a bug where the bar was not redrawn correctly in some multiscreen
          environments
        - add travis-ci support and make tests vastly more robust
    * config breakage
        - libqtile.layout.Stack's `stacks` parameter is now `num_stacks`
```

(continues on next page)

(continued from previous page)

```
Qtile 0.7.0, released 2014-03-30:
    * features
        - new disk free percentage widget
        - new widget to display static image
        - per core CPU graphs
        - add "screen affinity" in dynamic groups
        - volume widget changes volume linear-ly instead of log-ly
        - only draw bar when idle, vastly reducing the number of bar draws and
          speeding things up
        - new Gmail widget
        - Tile now supports automatically managing master windows via the
          `master_match` parameter.
        - include support for minimum height, width, size increment hints
    * bugfixes
        - don't crash on any exception in main loop
        - don't crash on exceptions in hooks
        - fix a ZeroDivisionError in CPU graph
        - remove a lot of duplicate and unused code
        - Steam windows are placed more correctly
        - Fixed several crashes in qsh
        - performance improvements for some layouts
        - keyboard layout widget behaves better with multiple keyboard
          configurations
    * config breakage
        - Tile's shuffleMatch is renamed to resetMaster

Qtile 0.6, released 2013-05-11:
    !!! Config breakage !!!
    This release breaks your config file in several ways:
        - The Textbox widget no longer takes a ``name'' positional parameter,
          since it was redundant; you can use the ``name'' kwarg to define it.
        - manager.Group (now _Group) is not used to configure groups any more;
          config.Group replaces it. For simple configurations (i.e.
          Group("a") type configs), this should be a drop in replacement.
          config.Group also provides many more options for showing and hiding
          groups, assigning windows to groups by default, etc.
        - The Key, Screen, Drag, and Click objects have moved from the manager
          module to the config module.
        - The Match object has moved from the dgroups module to the config
          module.
        - The addgroup hook now takes two parameters: the qtile object and the
          name of the group added:
              @hook.subscribe
              def addgroup_hook(qtile, name):
                  pass
        - The nextgroup and prevgroup commands are now on Screen instead of
          Group.
    For most people, you should be able to just:
        sed -i -e 's/libqtile.manager/libqtile.config' config.py
    ...dgroups users will need to go to a bit more work, but hopefully
    configuration will be much simpler now for new users.
    * features
```

(continues on next page)

(continued from previous page)

```
        - New widgets: task list,
        - New layout: Matrix
        - Added ability to drag and drop groups on GroupBox
        - added "next urgent window" command
        - added font shadowing on widgets
        - maildir widget supports multiple folders
        - new config option log_level to set logging level (any of
          logging.{DEBUG, INFO, WARNING, ERROR, CRITICAL})
        - add option to battery widget to hide while level is above a certain
          amount
        - vastly simplify configuration of dynamic groups
        - MPD widget now supports lots of metadata options
    * bugfixes
        - don't crash on restart when the config has errors
        - save layout and selected group state on restart
        - varous EWMH properties implemented correctly
        - fix non-black systray icon backgrounds
        - drastically reduce the number of timeout_add calls in most widgets
        - restart on RandR attach events to allow for new screens
        - log level defaults to ERROR
        - default config options are no longer initialized when users define
          their corresponding option (preventing duplicate widgets, etc.)
        - don't try to load config in qsh (not used)
        - fix font alignment across Textbox based widgets

Qtile 0.5, released 2012-11-11:
    (Note, this is not complete! Many, many changes have gone in to 0.5, by a
    large number of contributors. Thanks to everyone who reported a bug or
    fixed one!)
    * features
        - Test framework is now nose
        - Documentation is now in sphinx
        - Several install guides for various OSes
        - New widgets: battery based icon, MPRIS1, canto, current layout, yahoo
          weather, sensors, screen brightness, notifiy, pacman, windowtabs,
          she, crashme, wifi.
        - Several improvements to old widgets (e.g. battery widget displays low
          battery in red, GroupBox now has a better indication of which screen
          has focus in multi-screen setups, improvements to Prompt, etc.)
        - Desktop notification service.
        - More sane way to handle configuration files
        - Promote dgroups to a first class entity in libqtile
        - Allow layouts to be named on an instance level, so you can:
            layouts = [
                # a layout just for gimp
                layout.Slice('left', 192, name='gimp', role='gimp-toolbox',
                    fallback=layout.Slice('right', 256, role='gimp-dock',
                    fallback=layout.Stack(stacks=1, **border_args)))
            ]
            ...

            dynamic_groups = { 'gimp': {'layout': 'gimp'} }
```

(continues on next page)

(continued from previous page)

```
            Dgroups(..., dynamic_groups, ...)
        - New Layout: Zoomy
        - Add a session manager to re-exec qtile if things go south
        - Support for WM_TAKE_FOCUS protocol
        - Basic .desktop file for support in login managers
        - Qsh reconnects after qtile is restarted from within it
        - Textbox supports pango markup
        - Examples moved to qtile-examples repository.

    * bugfixes
        - Fix several classes of X races in a more sane way
        - Minor typo fixes to most widgets
        - Fix several crashes when drawing systray icons too early
        - Create directories for qtile socket as necessary
        - PEP8 formatting updates (though we're not totally there yet)
        - All unit tests pass
        - Lots of bugfixes to MonadTall
        - Create IPC socket directory if necessary
        - Better error if two widgets have STRETCH length
        - Autofloat window classes can now be overridden
        - xkeysyms updated

# vim :set ts=4 sw=4 sts=4 et :
```

INDEX

Symbols

_Group (*class in libqtile.group*), 266
_window_mask() (*libqtile.widget.systray.Systray method*), 341

A

add() (*libqtile.layout.matrix.Matrix method*), 234
add() (*libqtile.layout.stack.Stack method*), 251
add_rule() (*libqtile.core.manager.Qtile method*), 220
add_section() (*libqtile.layout.tree.TreeTab method*), 255
addgroup() (*libqtile.core.manager.Qtile method*), 220
addgroup() (*libqtile.hook.subscribe method*), 181
AGroupBox (*class in libqtile.widget.groupbox*), 271

B

Backlight (*class in libqtile.widget.backlight*), 273
Bar (*class in libqtile.bar*), 269
Battery (*class in libqtile.widget.battery*), 274
BatteryIcon (*class in libqtile.widget.battery*), 275
Bluetooth (*class in libqtile.widget.bluetooth*), 276
bring_to_front() (*libqtile.backend.base.window.Window method*), 261
Bsp (*class in libqtile.layout.bsp*), 227

C

Canto (*class in libqtile.widget.canto*), 280
CapsNumLockIndicator (*class in libqtile.widget.caps_num_lock_indicator*), 281
center() (*libqtile.backend.base.window.Window method*), 261
change_backlight() (*libqtile.widget.backlight.Backlight method*), 273
change_vt() (*libqtile.backend.wayland.core.Core method*), 361
change_window_order() (*libqtile.core.manager.Qtile method*), 220
changegroup() (*libqtile.hook.subscribe method*), 181
charge_dynamically() (*libqtile.widget.battery.Battery method*), 274
charge_to_full() (*libqtile.widget.battery.Battery method*), 274
CheckUpdates (*class in libqtile.widget.check_updates*), 282
Chord (*class in libqtile.widget.chord*), 283
clear() (*libqtile.widget.notify.Notify method*), 324
click() (*libqtile.widget.bluetooth.Bluetooth method*), 276
client_focus() (*libqtile.hook.subscribe method*), 181
client_killed() (*libqtile.hook.subscribe method*), 182
client_managed() (*libqtile.hook.subscribe method*), 182
client_mouse_enter() (*libqtile.hook.subscribe method*), 182
client_name_updated() (*libqtile.hook.subscribe method*), 182
client_new() (*libqtile.hook.subscribe method*), 183
client_to_next() (*libqtile.layout.stack.Stack method*), 251
client_to_previous() (*libqtile.layout.stack.Stack method*), 251
client_to_stack() (*libqtile.layout.stack.Stack method*), 251
client_urgent_hint_changed() (*libqtile.hook.subscribe method*), 183
Clipboard (*class in libqtile.widget.clipboard*), 284
Clock (*class in libqtile.widget.clock*), 285
close() (*libqtile.widget.widgetbox.WidgetBox method*), 350
Cmus (*class in libqtile.widget.cmus*), 287
collapse_branch() (*libqtile.layout.tree.TreeTab method*), 255
Columns (*class in libqtile.layout.columns*), 230
commands() (*libqtile.backend.base.window.Window method*), 262
commands() (*libqtile.backend.wayland.core.Core method*), 361
commands() (*libqtile.backend.x11.core.Core method*), 360
commands() (*libqtile.bar.Bar method*), 270
commands() (*libqtile.config.Screen method*), 358
commands() (*libqtile.core.manager.Qtile method*), 220

commands() (*libqtile.group._Group method*), 266
commands() (*libqtile.layout.bsp.Bsp method*), 228
commands() (*libqtile.layout.columns.Columns method*), 231
commands() (*libqtile.layout.floating.Floating method*), 233
commands() (*libqtile.layout.matrix.Matrix method*), 234
commands() (*libqtile.layout.max.Max method*), 236
commands() (*libqtile.layout.ratiotile.RatioTile method*), 245
commands() (*libqtile.layout.screensplit.ScreenSplit method*), 247
commands() (*libqtile.layout.slice.Slice method*), 248
commands() (*libqtile.layout.spiral.Spiral method*), 250
commands() (*libqtile.layout.stack.Stack method*), 251
commands() (*libqtile.layout.tile.Tile method*), 253
commands() (*libqtile.layout.tree.TreeTab method*), 255
commands() (*libqtile.layout.verticaltile.VerticalTile method*), 257
commands() (*libqtile.layout.xmonad.MonadTall method*), 237
commands() (*libqtile.layout.xmonad.MonadThreeCol method*), 240
commands() (*libqtile.layout.xmonad.MonadWide method*), 243
commands() (*libqtile.layout.zoomy.Zoomy method*), 259
commands() (*libqtile.widget.backlight.Backlight method*), 273
commands() (*libqtile.widget.base.Mirror method*), 317
commands() (*libqtile.widget.battery.Battery method*), 274
commands() (*libqtile.widget.battery.BatteryIcon method*), 275
commands() (*libqtile.widget.bluetooth.Bluetooth method*), 276
commands() (*libqtile.widget.canto.Canto method*), 280
commands() (*libqtile.widget.caps_num_lock_indicator.Caps method*), 281
commands() (*libqtile.widget.check_updates.CheckUpdates method*), 282
commands() (*libqtile.widget.chord.Chord method*), 283
commands() (*libqtile.widget.clipboard.Clipboard method*), 285
commands() (*libqtile.widget.clock.Clock method*), 286
commands() (*libqtile.widget.cmus.Cmus method*), 287
commands() (*libqtile.widget.countdown.Countdown method*), 288
commands() (*libqtile.widget.cpu.CPU method*), 278
commands() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 289
commands() (*libqtile.widget.currentlayout.CurrentLayout method*), 290
commands() (*libqtile.widget.currentlayout.CurrentLayoutIcon method*), 292
commands() (*libqtile.widget.currentscreen.CurrentScreen method*), 293
commands() (*libqtile.widget.df.DF method*), 294
commands() (*libqtile.widget.do_not_disturb.DoNotDisturb method*), 295
commands() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 296
commands() (*libqtile.widget.generic_poll_text.GenPollText method*), 297
commands() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
commands() (*libqtile.widget.gmail_checker.GmailChecker method*), 300
commands() (*libqtile.widget.graph.CPUGraph method*), 279
commands() (*libqtile.widget.graph.HDDBusyGraph method*), 302
commands() (*libqtile.widget.graph.HDDGraph method*), 303
commands() (*libqtile.widget.graph.MemoryGraph method*), 316
commands() (*libqtile.widget.graph.NetGraph method*), 323
commands() (*libqtile.widget.graph.SwapGraph method*), 340
commands() (*libqtile.widget.groupbox.AGroupBox method*), 272
commands() (*libqtile.widget.groupbox.GroupBox method*), 301
commands() (*libqtile.widget.idlerpg.IdleRPG method*), 304
commands() (*libqtile.widget.image.Image method*), 305
commands() (*libqtile.widget.imapwidget.ImapWidget method*), 306
commands() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 307
commands() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
commands() (*libqtile.widget.khal_calendar.KhalCalendar method*), 310
commands() (*libqtile.widget.launchbar.LaunchBar method*), 311
commands() (*libqtile.widget.load.Load method*), 312
commands() (*libqtile.widget.maildir.Maildir method*), 313
commands() (*libqtile.widget.memory.Memory method*), 315
commands() (*libqtile.widget.moc.Moc method*), 318
commands() (*libqtile.widget.mpd2widget.Mpd2 method*), 319
commands() (*libqtile.widget.mpris2widget.Mpris2 method*), 320
commands() (*libqtile.widget.net.Net method*), 321
commands() (*libqtile.widget.notify.Notify method*), 324

commands() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 325
commands() (*libqtile.widget.open_weather.OpenWeather method*), 326
commands() (*libqtile.widget.pomodoro.Pomodoro method*), 328
commands() (*libqtile.widget.prompt.Prompt method*), 329
commands() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
commands() (*libqtile.widget.quick_exit.QuickExit method*), 332
commands() (*libqtile.widget.screensplit.ScreenSplit method*), 333
commands() (*libqtile.widget.sensors.ThermalSensor method*), 345
commands() (*libqtile.widget.sep.Sep method*), 334
commands() (*libqtile.widget.she.She method*), 335
commands() (*libqtile.widget.spacer.Spacer method*), 337
commands() (*libqtile.widget.statusnotifier.StatusNotifier method*), 337
commands() (*libqtile.widget.stock_ticker.StockTicker method*), 339
commands() (*libqtile.widget.systray.Systray method*), 341
commands() (*libqtile.widget.tasklist.TaskList method*), 343
commands() (*libqtile.widget.textbox.TextBox method*), 344
commands() (*libqtile.widget.thermal_zone.ThermalZone method*), 346
commands() (*libqtile.widget.volume.Volume method*), 347
commands() (*libqtile.widget.wallpaper.Wallpaper method*), 349
commands() (*libqtile.widget.widgetbox.WidgetBox method*), 350
commands() (*libqtile.widget.window_count.WindowCount method*), 351
commands() (*libqtile.widget.windowname.WindowName method*), 352
commands() (*libqtile.widget.windowtabs.WindowTabs method*), 353
commands() (*libqtile.widget.wlan.Wlan method*), 354
commands() (*libqtile.widget.wttr.Wttr method*), 356
Core (*class in libqtile.backend.wayland.core*), 361
Core (*class in libqtile.backend.x11.core*), 360
Countdown (*class in libqtile.widget.countdown*), 288
CPU (*class in libqtile.widget.cpu*), 277
CPUGraph (*class in libqtile.widget.graph*), 279
critical() (*libqtile.core.manager.Qtile method*), 220
CryptoTicker (*class in libqtile.widget.crypto_ticker*), 289
current_screen_change() (*libqtile.hook.subscribe method*), 183
CurrentLayout (*class in libqtile.widget.currentlayout*), 290
CurrentLayoutIcon (*class in libqtile.widget.currentlayout*), 291
CurrentScreen (*class in libqtile.widget.currentscreen*), 292

D

debug() (*libqtile.core.manager.Qtile method*), 220
decrease_nmaster() (*libqtile.layout.tile.Tile method*), 253
decrease_ratio() (*libqtile.layout.ratiotile.RatioTile method*), 246
decrease_ratio() (*libqtile.layout.spiral.Spiral method*), 250
decrease_ratio() (*libqtile.layout.tile.Tile method*), 253
decrease_ratio() (*libqtile.layout.tree.TreeTab method*), 255
decrease_vol() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
decrease_vol() (*libqtile.widget.volume.Volume method*), 348
del_section() (*libqtile.layout.tree.TreeTab method*), 255
delete() (*libqtile.layout.matrix.Matrix method*), 234
delete() (*libqtile.layout.stack.Stack method*), 251
delgroup() (*libqtile.core.manager.Qtile method*), 220
delgroup() (*libqtile.hook.subscribe method*), 184
DF (*class in libqtile.widget.df*), 294
disable_floating() (*libqtile.backend.base.window.Window method*), 262
disable_fullscreen() (*libqtile.backend.base.window.Window method*), 262
display() (*libqtile.widget.notify.Notify method*), 324
display_kb() (*libqtile.core.manager.Qtile method*), 221
doc() (*libqtile.backend.base.window.Window method*), 262
doc() (*libqtile.backend.wayland.core.Core method*), 361
doc() (*libqtile.backend.x11.core.Core method*), 360
doc() (*libqtile.bar.Bar method*), 270
doc() (*libqtile.config.Screen method*), 358
doc() (*libqtile.core.manager.Qtile method*), 221
doc() (*libqtile.group._Group method*), 266
doc() (*libqtile.layout.bsp.Bsp method*), 229
doc() (*libqtile.layout.columns.Columns method*), 231
doc() (*libqtile.layout.floating.Floating method*), 233
doc() (*libqtile.layout.matrix.Matrix method*), 234
doc() (*libqtile.layout.max.Max method*), 236
doc() (*libqtile.layout.ratiotile.RatioTile method*), 246
doc() (*libqtile.layout.screensplit.ScreenSplit method*), 247

doc() (*libqtile.layout.slice.Slice method*), 248
doc() (*libqtile.layout.spiral.Spiral method*), 250
doc() (*libqtile.layout.stack.Stack method*), 252
doc() (*libqtile.layout.tile.Tile method*), 253
doc() (*libqtile.layout.tree.TreeTab method*), 255
doc() (*libqtile.layout.verticaltile.VerticalTile method*), 257
doc() (*libqtile.layout.xmonad.MonadTall method*), 237
doc() (*libqtile.layout.xmonad.MonadThreeCol method*), 240
doc() (*libqtile.layout.xmonad.MonadWide method*), 243
doc() (*libqtile.layout.zoomy.Zoomy method*), 259
doc() (*libqtile.widget.backlight.Backlight method*), 273
doc() (*libqtile.widget.base.Mirror method*), 317
doc() (*libqtile.widget.battery.Battery method*), 274
doc() (*libqtile.widget.battery.BatteryIcon method*), 275
doc() (*libqtile.widget.bluetooth.Bluetooth method*), 277
doc() (*libqtile.widget.canto.Canto method*), 280
doc() (*libqtile.widget.caps_num_lock_indicator.CapsNumLockIndicator method*), 281
doc() (*libqtile.widget.check_updates.CheckUpdates method*), 282
doc() (*libqtile.widget.chord.Chord method*), 284
doc() (*libqtile.widget.clipboard.Clipboard method*), 285
doc() (*libqtile.widget.clock.Clock method*), 286
doc() (*libqtile.widget.cmus.Cmus method*), 287
doc() (*libqtile.widget.countdown.Countdown method*), 288
doc() (*libqtile.widget.cpu.CPU method*), 278
doc() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 289
doc() (*libqtile.widget.currentlayout.CurrentLayout method*), 291
doc() (*libqtile.widget.currentlayout.CurrentLayoutIcon method*), 292
doc() (*libqtile.widget.currentscreen.CurrentScreen method*), 293
doc() (*libqtile.widget.df.DF method*), 294
doc() (*libqtile.widget.do_not_disturb.DoNotDisturb method*), 295
doc() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 296
doc() (*libqtile.widget.generic_poll_text.GenPollText method*), 297
doc() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
doc() (*libqtile.widget.gmail_checker.GmailChecker method*), 300
doc() (*libqtile.widget.graph.CPUGraph method*), 279
doc() (*libqtile.widget.graph.HDDBusyGraph method*), 302
doc() (*libqtile.widget.graph.HDDGraph method*), 303
doc() (*libqtile.widget.graph.MemoryGraph method*), 316
doc() (*libqtile.widget.graph.NetGraph method*), 323
doc() (*libqtile.widget.graph.SwapGraph method*), 340
doc() (*libqtile.widget.groupbox.AGroupBox method*), 272
doc() (*libqtile.widget.groupbox.GroupBox method*), 301
doc() (*libqtile.widget.idlerpg.IdleRPG method*), 304
doc() (*libqtile.widget.image.Image method*), 305
doc() (*libqtile.widget.imapwidget.ImapWidget method*), 306
doc() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 308
doc() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
doc() (*libqtile.widget.khal_calendar.KhalCalendar method*), 310
doc() (*libqtile.widget.launchbar.LaunchBar method*), 311
doc() (*libqtile.widget.load.Load method*), 312
doc() (*libqtile.widget.maildir.Maildir method*), 313
doc() (*libqtile.widget.memory.Memory method*), 315
doc() (*libqtile.widget.moc.Moc method*), 318
doc() (*libqtile.widget.mpd2widget.Mpd2 method*), 319
doc() (*libqtile.widget.mpris2widget.Mpris2 method*), 320
doc() (*libqtile.widget.net.Net method*), 322
doc() (*libqtile.widget.notify.Notify method*), 324
doc() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 325
doc() (*libqtile.widget.open_weather.OpenWeather method*), 326
doc() (*libqtile.widget.pomodoro.Pomodoro method*), 328
doc() (*libqtile.widget.prompt.Prompt method*), 329
doc() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
doc() (*libqtile.widget.quick_exit.QuickExit method*), 332
doc() (*libqtile.widget.screensplit.ScreenSplit method*), 333
doc() (*libqtile.widget.sensors.ThermalSensor method*), 345
doc() (*libqtile.widget.sep.Sep method*), 335
doc() (*libqtile.widget.she.She method*), 336
doc() (*libqtile.widget.spacer.Spacer method*), 337
doc() (*libqtile.widget.statusnotifier.StatusNotifier method*), 338
doc() (*libqtile.widget.stock_ticker.StockTicker method*), 339
doc() (*libqtile.widget.systray.Systray method*), 341
doc() (*libqtile.widget.tasklist.TaskList method*), 343
doc() (*libqtile.widget.textbox.TextBox method*), 344
doc() (*libqtile.widget.thermal_zone.ThermalZone method*), 346
doc() (*libqtile.widget.volume.Volume method*), 348
doc() (*libqtile.widget.wallpaper.Wallpaper method*), 349
doc() (*libqtile.widget.widgetbox.WidgetBox method*),

350
doc() (*libqtile.widget.window_count.WindowCount method*), 351
doc() (*libqtile.widget.windowname.WindowName method*), 352
doc() (*libqtile.widget.windowtabs.WindowTabs method*), 353
doc() (*libqtile.widget.wlan.Wlan method*), 355
doc() (*libqtile.widget.wttr.Wttr method*), 356
DoNotDisturb (*class in libqtile.widget.do_not_disturb*), 295
down() (*libqtile.layout.bsp.Bsp method*), 229
down() (*libqtile.layout.columns.Columns method*), 232
down() (*libqtile.layout.matrix.Matrix method*), 235
down() (*libqtile.layout.max.Max method*), 236
down() (*libqtile.layout.stack.Stack method*), 252
down_opacity() (*libqtile.backend.base.window.Window method*), 262

E

enable_floating() (*libqtile.backend.base.window.Window method*), 262
enable_fullscreen() (*libqtile.backend.base.window.Window method*), 262
enter_chord() (*libqtile.hook.subscribe method*), 184
error() (*libqtile.core.manager.Qtile method*), 221
eval() (*libqtile.backend.base.window.Window method*), 262
eval() (*libqtile.backend.wayland.core.Core method*), 361
eval() (*libqtile.backend.x11.core.Core method*), 360
eval() (*libqtile.bar.Bar method*), 270
eval() (*libqtile.config.Screen method*), 358
eval() (*libqtile.core.manager.Qtile method*), 221
eval() (*libqtile.group._Group method*), 266
eval() (*libqtile.layout.bsp.Bsp method*), 229
eval() (*libqtile.layout.columns.Columns method*), 232
eval() (*libqtile.layout.floating.Floating method*), 233
eval() (*libqtile.layout.matrix.Matrix method*), 235
eval() (*libqtile.layout.max.Max method*), 236
eval() (*libqtile.layout.ratiotile.RatioTile method*), 246
eval() (*libqtile.layout.screensplit.ScreenSplit method*), 247
eval() (*libqtile.layout.slice.Slice method*), 248
eval() (*libqtile.layout.spiral.Spiral method*), 250
eval() (*libqtile.layout.stack.Stack method*), 252
eval() (*libqtile.layout.tile.Tile method*), 253
eval() (*libqtile.layout.tree.TreeTab method*), 255
eval() (*libqtile.layout.verticaltile.VerticalTile method*), 257
eval() (*libqtile.layout.xmonad.MonadTall method*), 238
eval() (*libqtile.layout.xmonad.MonadThreeCol method*), 240
eval() (*libqtile.layout.xmonad.MonadWide method*), 243
eval() (*libqtile.layout.zoomy.Zoomy method*), 259
eval() (*libqtile.widget.backlight.Backlight method*), 273
eval() (*libqtile.widget.base.Mirror method*), 317
eval() (*libqtile.widget.battery.Battery method*), 274
eval() (*libqtile.widget.battery.BatteryIcon method*), 275
eval() (*libqtile.widget.bluetooth.Bluetooth method*), 277
eval() (*libqtile.widget.canto.Canto method*), 280
eval() (*libqtile.widget.caps_num_lock_indicator.CapsNumLockIndicator method*), 281
eval() (*libqtile.widget.check_updates.CheckUpdates method*), 282
eval() (*libqtile.widget.chord.Chord method*), 284
eval() (*libqtile.widget.clipboard.Clipboard method*), 285
eval() (*libqtile.widget.clock.Clock method*), 286
eval() (*libqtile.widget.cmus.Cmus method*), 287
eval() (*libqtile.widget.countdown.Countdown method*), 288
eval() (*libqtile.widget.cpu.CPU method*), 278
eval() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 289
eval() (*libqtile.widget.currentlayout.CurrentLayout method*), 291
eval() (*libqtile.widget.currentlayout.CurrentLayoutIcon method*), 292
eval() (*libqtile.widget.currentscreen.CurrentScreen method*), 293
eval() (*libqtile.widget.df.DF method*), 294
eval() (*libqtile.widget.do_not_disturb.DoNotDisturb method*), 295
eval() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 296
eval() (*libqtile.widget.generic_poll_text.GenPollText method*), 298
eval() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
eval() (*libqtile.widget.gmail_checker.GmailChecker method*), 300
eval() (*libqtile.widget.graph.CPUGraph method*), 279
eval() (*libqtile.widget.graph.HDDBusyGraph method*), 302
eval() (*libqtile.widget.graph.HDDGraph method*), 303
eval() (*libqtile.widget.graph.MemoryGraph method*), 316
eval() (*libqtile.widget.graph.NetGraph method*), 323
eval() (*libqtile.widget.graph.SwapGraph method*), 340
eval() (*libqtile.widget.groupbox.AGroupBox method*), 272
eval() (*libqtile.widget.groupbox.GroupBox method*), 301
eval() (*libqtile.widget.idlerpg.IdleRPG method*), 304
eval() (*libqtile.widget.image.Image method*), 305

eval() (*libqtile.widget.imapwidget.ImapWidget method*), 306
eval() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 308
eval() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
eval() (*libqtile.widget.khal_calendar.KhalCalendar method*), 310
eval() (*libqtile.widget.launchbar.LaunchBar method*), 311
eval() (*libqtile.widget.load.Load method*), 312
eval() (*libqtile.widget.maildir.Maildir method*), 314
eval() (*libqtile.widget.memory.Memory method*), 315
eval() (*libqtile.widget.moc.Moc method*), 318
eval() (*libqtile.widget.mpd2widget.Mpd2 method*), 319
eval() (*libqtile.widget.mpris2widget.Mpris2 method*), 320
eval() (*libqtile.widget.net.Net method*), 322
eval() (*libqtile.widget.notify.Notify method*), 324
eval() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 325
eval() (*libqtile.widget.open_weather.OpenWeather method*), 327
eval() (*libqtile.widget.pomodoro.Pomodoro method*), 328
eval() (*libqtile.widget.prompt.Prompt method*), 329
eval() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
eval() (*libqtile.widget.quick_exit.QuickExit method*), 332
eval() (*libqtile.widget.screensplit.ScreenSplit method*), 333
eval() (*libqtile.widget.sensors.ThermalSensor method*), 345
eval() (*libqtile.widget.sep.Sep method*), 335
eval() (*libqtile.widget.she.She method*), 336
eval() (*libqtile.widget.spacer.Spacer method*), 337
eval() (*libqtile.widget.statusnotifier.StatusNotifier method*), 338
eval() (*libqtile.widget.stock_ticker.StockTicker method*), 339
eval() (*libqtile.widget.systray.Systray method*), 341
eval() (*libqtile.widget.tasklist.TaskList method*), 343
eval() (*libqtile.widget.textbox.TextBox method*), 344
eval() (*libqtile.widget.thermal_zone.ThermalZone method*), 346
eval() (*libqtile.widget.volume.Volume method*), 348
eval() (*libqtile.widget.wallpaper.Wallpaper method*), 349
eval() (*libqtile.widget.widgetbox.WidgetBox method*), 350
eval() (*libqtile.widget.window_count.WindowCount method*), 351
eval() (*libqtile.widget.windowname.WindowName method*), 352
eval() (*libqtile.widget.windowtabs.WindowTabs method*), 354
eval() (*libqtile.widget.wlan.Wlan method*), 355
eval() (*libqtile.widget.wttr.Wttr method*), 356
exec_general() (*libqtile.widget.prompt.Prompt method*), 329
expand_branch() (*libqtile.layout.tree.TreeTab method*), 256

F

fake_button_press() (*libqtile.bar.Bar method*), 270
fake_keypress() (*libqtile.widget.prompt.Prompt method*), 330
findwindow() (*libqtile.core.manager.Qtile method*), 221
fire_user_hook() (*libqtile.core.manager.Qtile method*), 221
flip() (*libqtile.layout.xmonad.MonadTall method*), 238
flip() (*libqtile.layout.xmonad.MonadThreeCol method*), 240
flip() (*libqtile.layout.xmonad.MonadWide method*), 243
flip_down() (*libqtile.layout.bsp.Bsp method*), 229
flip_left() (*libqtile.layout.bsp.Bsp method*), 229
flip_right() (*libqtile.layout.bsp.Bsp method*), 229
flip_up() (*libqtile.layout.bsp.Bsp method*), 229
float_change() (*libqtile.hook.subscribe method*), 184
Floating (*class in libqtile.layout.floating*), 233
focus() (*libqtile.backend.base.window.Window method*), 262
focus() (*libqtile.widget.systray.Systray method*), 342
focus_back() (*libqtile.group._Group method*), 267
focus_by_index() (*libqtile.group._Group method*), 267
focus_by_name() (*libqtile.group._Group method*), 267
focus_change() (*libqtile.hook.subscribe method*), 185
force_update() (*libqtile.widget.battery.Battery method*), 274
force_update() (*libqtile.widget.canto.Canto method*), 280
force_update() (*libqtile.widget.caps_num_lock_indicator.CapsNumLock method*), 281
force_update() (*libqtile.widget.check_updates.CheckUpdates method*), 283
force_update() (*libqtile.widget.cmus.Cmus method*), 287
force_update() (*libqtile.widget.cpu.CPU method*), 278
force_update() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 290
force_update() (*libqtile.widget.df.DF method*), 294
force_update() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 296
force_update() (*libqtile.widget.generic_poll_text.GenPollText method*), 298

force_update() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
force_update() (*libqtile.widget.gmail_checker.GmailChecker method*), 300
force_update() (*libqtile.widget.idlerpg.IdleRPG method*), 304
force_update() (*libqtile.widget.imapwidget.ImapWidget method*), 307
force_update() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 308
force_update() (*libqtile.widget.khal_calendar.KhalCalendar method*), 310
force_update() (*libqtile.widget.load.Load method*), 312
force_update() (*libqtile.widget.maildir.Maildir method*), 314
force_update() (*libqtile.widget.memory.Memory method*), 315
force_update() (*libqtile.widget.moc.Moc method*), 318
force_update() (*libqtile.widget.mpd2widget.Mpd2 method*), 319
force_update() (*libqtile.widget.net.Net method*), 322
force_update() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 325
force_update() (*libqtile.widget.open_weather.OpenWeather method*), 327
force_update() (*libqtile.widget.pomodoro.Pomodoro method*), 328
force_update() (*libqtile.widget.stock_ticker.StockTicker method*), 339
force_update() (*libqtile.widget.thermal_zone.ThermalZone method*), 346
force_update() (*libqtile.widget.wttr.Wttr method*), 356
function() (*libqtile.backend.base.window.Window method*), 262
function() (*libqtile.backend.wayland.core.Core method*), 361
function() (*libqtile.backend.x11.core.Core method*), 360
function() (*libqtile.bar.Bar method*), 270
function() (*libqtile.config.Screen method*), 358
function() (*libqtile.core.manager.Qtile method*), 221
function() (*libqtile.group._Group method*), 267
function() (*libqtile.layout.bsp.Bsp method*), 229
function() (*libqtile.layout.columns.Columns method*), 232
function() (*libqtile.layout.floating.Floating method*), 233
function() (*libqtile.layout.matrix.Matrix method*), 235
function() (*libqtile.layout.max.Max method*), 236
function() (*libqtile.layout.ratiotile.RatioTile method*), 246
function() (*libqtile.layout.screensplit.ScreenSplit method*), 247
function() (*libqtile.layout.slice.Slice method*), 248
function() (*libqtile.layout.spiral.Spiral method*), 250
function() (*libqtile.layout.stack.Stack method*), 252
function() (*libqtile.layout.tile.Tile method*), 254
function() (*libqtile.layout.tree.TreeTab method*), 256
function() (*libqtile.layout.verticaltile.VerticalTile method*), 257
function() (*libqtile.layout.xmonad.MonadTall method*), 238
function() (*libqtile.layout.xmonad.MonadThreeCol method*), 240
function() (*libqtile.layout.xmonad.MonadWide method*), 243
function() (*libqtile.layout.zoomy.Zoomy method*), 259
function() (*libqtile.widget.backlight.Backlight method*), 273
function() (*libqtile.widget.base.Mirror method*), 317
function() (*libqtile.widget.battery.Battery method*), 275
function() (*libqtile.widget.battery.BatteryIcon method*), 276
function() (*libqtile.widget.bluetooth.Bluetooth method*), 277
function() (*libqtile.widget.canto.Canto method*), 280
function() (*libqtile.widget.caps_num_lock_indicator.CapsNumLockIndic method*), 281
function() (*libqtile.widget.check_updates.CheckUpdates method*), 283
function() (*libqtile.widget.chord.Chord method*), 284
function() (*libqtile.widget.clipboard.Clipboard method*), 285
function() (*libqtile.widget.clock.Clock method*), 286
function() (*libqtile.widget.cmus.Cmus method*), 287
function() (*libqtile.widget.countdown.Countdown method*), 288
function() (*libqtile.widget.cpu.CPU method*), 278
function() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 290
function() (*libqtile.widget.currentlayout.CurrentLayout method*), 291
function() (*libqtile.widget.currentlayout.CurrentLayoutIcon method*), 292
function() (*libqtile.widget.currentscreen.CurrentScreen method*), 293
function() (*libqtile.widget.df.DF method*), 294
function() (*libqtile.widget.do_not_disturb.DoNotDisturb method*), 295
function() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 297
function() (*libqtile.widget.generic_poll_text.GenPollText method*), 298
function() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
function() (*libqtile.widget.gmail_checker.GmailChecker*

method), 300
function() (*libqtile.widget.graph.CPUGraph method*), 279
function() (*libqtile.widget.graph.HDDBusyGraph method*), 302
function() (*libqtile.widget.graph.HDDGraph method*), 303
function() (*libqtile.widget.graph.MemoryGraph method*), 316
function() (*libqtile.widget.graph.NetGraph method*), 323
function() (*libqtile.widget.graph.SwapGraph method*), 340
function() (*libqtile.widget.groupbox.AGroupBox method*), 272
function() (*libqtile.widget.groupbox.GroupBox method*), 301
function() (*libqtile.widget.idlerpg.IdleRPG method*), 304
function() (*libqtile.widget.image.Image method*), 305
function() (*libqtile.widget.imapwidget.ImapWidget method*), 307
function() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 308
function() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
function() (*libqtile.widget.khal_calendar.KhalCalendar method*), 310
function() (*libqtile.widget.launchbar.LaunchBar method*), 311
function() (*libqtile.widget.load.Load method*), 313
function() (*libqtile.widget.maildir.Maildir method*), 314
function() (*libqtile.widget.memory.Memory method*), 315
function() (*libqtile.widget.moc.Moc method*), 318
function() (*libqtile.widget.mpd2widget.Mpd2 method*), 319
function() (*libqtile.widget.mpris2widget.Mpris2 method*), 320
function() (*libqtile.widget.net.Net method*), 322
function() (*libqtile.widget.notify.Notify method*), 324
function() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 326
function() (*libqtile.widget.open_weather.OpenWeather method*), 327
function() (*libqtile.widget.pomodoro.Pomodoro method*), 328
function() (*libqtile.widget.prompt.Prompt method*), 330
function() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
function() (*libqtile.widget.quick_exit.QuickExit method*), 332
function() (*libqtile.widget.screensplit.ScreenSplit method*), 334
function() (*libqtile.widget.sensors.ThermalSensor method*), 345
function() (*libqtile.widget.sep.Sep method*), 335
function() (*libqtile.widget.she.She method*), 336
function() (*libqtile.widget.spacer.Spacer method*), 337
function() (*libqtile.widget.statusnotifier.StatusNotifier method*), 338
function() (*libqtile.widget.stock_ticker.StockTicker method*), 339
function() (*libqtile.widget.systray.Systray method*), 342
function() (*libqtile.widget.tasklist.TaskList method*), 343
function() (*libqtile.widget.textbox.TextBox method*), 344
function() (*libqtile.widget.thermal_zone.ThermalZone method*), 347
function() (*libqtile.widget.volume.Volume method*), 348
function() (*libqtile.widget.wallpaper.Wallpaper method*), 349
function() (*libqtile.widget.widgetbox.WidgetBox method*), 350
function() (*libqtile.widget.window_count.WindowCount method*), 351
function() (*libqtile.widget.windowname.WindowName method*), 353
function() (*libqtile.widget.windowtabs.WindowTabs method*), 354
function() (*libqtile.widget.wlan.Wlan method*), 355
function() (*libqtile.widget.wttr.Wttr method*), 356

G

GenPollCommand (*class in libqtile.widget.generic_poll_text*), 296
GenPollText (*class in libqtile.widget.generic_poll_text*), 297
GenPollUrl (*class in libqtile.widget.generic_poll_text*), 298
get() (*libqtile.widget.screensplit.ScreenSplit method*), 334
get() (*libqtile.widget.textbox.TextBox method*), 344
get() (*libqtile.widget.window_count.WindowCount method*), 352
get_groups() (*libqtile.core.manager.Qtile method*), 221
get_hints() (*libqtile.widget.systray.Systray method*), 342
get_inputs() (*libqtile.backend.wayland.core.Core method*), 361
get_position() (*libqtile.backend.base.window.Window method*), 262

get_screens() (*libqtile.core.manager.Qtile method*), 221
get_size() (*libqtile.backend.base.window.Window method*), 262
get_state() (*libqtile.core.manager.Qtile method*), 221
get_test_data() (*libqtile.core.manager.Qtile method*), 221
GmailChecker (*class in libqtile.widget.gmail_checker*), 299
group_window_add() (*libqtile.hook.subscribe method*), 185
GroupBox (*class in libqtile.widget.groupbox*), 301
grow() (*libqtile.layout.verticaltile.VerticalTile method*), 258
grow() (*libqtile.layout.xmonad.MonadTall method*), 238
grow() (*libqtile.layout.xmonad.MonadThreeCol method*), 240
grow() (*libqtile.layout.xmonad.MonadWide method*), 244
grow_down() (*libqtile.layout.bsp.Bsp method*), 229
grow_down() (*libqtile.layout.columns.Columns method*), 232
grow_left() (*libqtile.layout.bsp.Bsp method*), 229
grow_left() (*libqtile.layout.columns.Columns method*), 232
grow_main() (*libqtile.layout.spiral.Spiral method*), 250
grow_main() (*libqtile.layout.xmonad.MonadTall method*), 238
grow_main() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
grow_main() (*libqtile.layout.xmonad.MonadWide method*), 244
grow_right() (*libqtile.layout.bsp.Bsp method*), 229
grow_right() (*libqtile.layout.columns.Columns method*), 232
grow_up() (*libqtile.layout.bsp.Bsp method*), 229
grow_up() (*libqtile.layout.columns.Columns method*), 232

H

HDDBusyGraph (*class in libqtile.widget.graph*), 302
HDDGraph (*class in libqtile.widget.graph*), 303
hide_cursor() (*libqtile.backend.wayland.core.Core method*), 361
hide_show_bar() (*libqtile.core.manager.Qtile method*), 221

I

IdleRPG (*class in libqtile.widget.idlerpg*), 304
Image (*class in libqtile.widget.image*), 305
ImapWidget (*class in libqtile.widget.imapwidget*), 306
increase_nmaster() (*libqtile.layout.tile.Tile method*), 254
increase_ratio() (*libqtile.layout.ratiotile.RatioTile method*), 246
increase_ratio() (*libqtile.layout.spiral.Spiral method*), 250
increase_ratio() (*libqtile.layout.tile.Tile method*), 254
increase_ratio() (*libqtile.layout.tree.TreeTab method*), 256
increase_vol() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
increase_vol() (*libqtile.widget.volume.Volume method*), 348
info() (*libqtile.backend.base.window.Window method*), 262
info() (*libqtile.backend.wayland.core.Core method*), 361
info() (*libqtile.backend.x11.core.Core method*), 360
info() (*libqtile.bar.Bar method*), 270
info() (*libqtile.config.Screen method*), 358
info() (*libqtile.core.manager.Qtile method*), 221
info() (*libqtile.group._Group method*), 267
info() (*libqtile.layout.bsp.Bsp method*), 229
info() (*libqtile.layout.columns.Columns method*), 232
info() (*libqtile.layout.floating.Floating method*), 233
info() (*libqtile.layout.matrix.Matrix method*), 235
info() (*libqtile.layout.max.Max method*), 236
info() (*libqtile.layout.ratiotile.RatioTile method*), 246
info() (*libqtile.layout.screensplit.ScreenSplit method*), 247
info() (*libqtile.layout.slice.Slice method*), 248
info() (*libqtile.layout.stack.Stack method*), 252
info() (*libqtile.layout.tile.Tile method*), 254
info() (*libqtile.layout.tree.TreeTab method*), 256
info() (*libqtile.layout.verticaltile.VerticalTile method*), 258
info() (*libqtile.layout.xmonad.MonadTall method*), 238
info() (*libqtile.layout.xmonad.MonadWide method*), 244
info() (*libqtile.layout.zoomy.Zoomy method*), 259
info() (*libqtile.widget.backlight.Backlight method*), 273
info() (*libqtile.widget.base.Mirror method*), 317
info() (*libqtile.widget.battery.Battery method*), 275
info() (*libqtile.widget.battery.BatteryIcon method*), 276
info() (*libqtile.widget.bluetooth.Bluetooth method*), 277
info() (*libqtile.widget.canto.Canto method*), 280
info() (*libqtile.widget.caps_num_lock_indicator.CapsNumLockIndicator method*), 281
info() (*libqtile.widget.check_updates.CheckUpdates method*), 283
info() (*libqtile.widget.chord.Chord method*), 284
info() (*libqtile.widget.clipboard.Clipboard method*), 285
info() (*libqtile.widget.clock.Clock method*), 286
info() (*libqtile.widget.cmus.Cmus method*), 287

info() (*libqtile.widget.countdown.Countdown method*), 289
info() (*libqtile.widget.cpu.CPU method*), 278
info() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 290
info() (*libqtile.widget.currentlayout.CurrentLayout method*), 291
info() (*libqtile.widget.currentlayout.CurrentLayoutIcon method*), 292
info() (*libqtile.widget.currentscreen.CurrentScreen method*), 293
info() (*libqtile.widget.df.DF method*), 294
info() (*libqtile.widget.do_not_disturb.DoNotDisturb method*), 295
info() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 297
info() (*libqtile.widget.generic_poll_text.GenPollText method*), 298
info() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
info() (*libqtile.widget.gmail_checker.GmailChecker method*), 300
info() (*libqtile.widget.graph.CPUGraph method*), 279
info() (*libqtile.widget.graph.HDDBusyGraph method*), 302
info() (*libqtile.widget.graph.HDDGraph method*), 303
info() (*libqtile.widget.graph.MemoryGraph method*), 316
info() (*libqtile.widget.graph.NetGraph method*), 323
info() (*libqtile.widget.graph.SwapGraph method*), 340
info() (*libqtile.widget.groupbox.AGroupBox method*), 272
info() (*libqtile.widget.groupbox.GroupBox method*), 301
info() (*libqtile.widget.idlerpg.IdleRPG method*), 304
info() (*libqtile.widget.image.Image method*), 306
info() (*libqtile.widget.imapwidget.ImapWidget method*), 307
info() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 308
info() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
info() (*libqtile.widget.khal_calendar.KhalCalendar method*), 310
info() (*libqtile.widget.launchbar.LaunchBar method*), 311
info() (*libqtile.widget.load.Load method*), 313
info() (*libqtile.widget.maildir.Maildir method*), 314
info() (*libqtile.widget.memory.Memory method*), 315
info() (*libqtile.widget.moc.Moc method*), 318
info() (*libqtile.widget.mpd2widget.Mpd2 method*), 319
info() (*libqtile.widget.mpris2widget.Mpris2 method*), 320
info() (*libqtile.widget.net.Net method*), 322
info() (*libqtile.widget.notify.Notify method*), 324
info() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 326
info() (*libqtile.widget.open_weather.OpenWeather method*), 327
info() (*libqtile.widget.pomodoro.Pomodoro method*), 328
info() (*libqtile.widget.prompt.Prompt method*), 330
info() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
info() (*libqtile.widget.quick_exit.QuickExit method*), 332
info() (*libqtile.widget.screensplit.ScreenSplit method*), 334
info() (*libqtile.widget.sensors.ThermalSensor method*), 345
info() (*libqtile.widget.sep.Sep method*), 335
info() (*libqtile.widget.she.She method*), 336
info() (*libqtile.widget.spacer.Spacer method*), 337
info() (*libqtile.widget.statusnotifier.StatusNotifier method*), 338
info() (*libqtile.widget.stock_ticker.StockTicker method*), 339
info() (*libqtile.widget.tasklist.TaskList method*), 343
info() (*libqtile.widget.textbox.TextBox method*), 344
info() (*libqtile.widget.thermal_zone.ThermalZone method*), 347
info() (*libqtile.widget.volume.Volume method*), 348
info() (*libqtile.widget.wallpaper.Wallpaper method*), 349
info() (*libqtile.widget.widgetbox.WidgetBox method*), 350
info() (*libqtile.widget.window_count.WindowCount method*), 352
info() (*libqtile.widget.windowname.WindowName method*), 353
info() (*libqtile.widget.windowtabs.WindowTabs method*), 354
info() (*libqtile.widget.wlan.Wlan method*), 355
info() (*libqtile.widget.wttr.Wttr method*), 356
info_by_name() (*libqtile.group._Group method*), 267
inspect() (*libqtile.widget.systray.Systray method*), 342
internal_windows() (*libqtile.core.manager.Qtile method*), 222
invoke() (*libqtile.widget.notify.Notify method*), 324
is_visible() (*libqtile.backend.base.window.Window method*), 262
items() (*libqtile.backend.base.window.Window method*), 262
items() (*libqtile.backend.wayland.core.Core method*), 362
items() (*libqtile.backend.x11.core.Core method*), 360
items() (*libqtile.bar.Bar method*), 270
items() (*libqtile.config.Screen method*), 358

items() (*libqtile.core.manager.Qtile method*), 222
items() (*libqtile.group._Group method*), 267
items() (*libqtile.layout.bsp.Bsp method*), 229
items() (*libqtile.layout.columns.Columns method*), 232
items() (*libqtile.layout.floating.Floating method*), 233
items() (*libqtile.layout.matrix.Matrix method*), 235
items() (*libqtile.layout.max.Max method*), 236
items() (*libqtile.layout.ratiotile.RatioTile method*), 246
items() (*libqtile.layout.screensplit.ScreenSplit method*), 247
items() (*libqtile.layout.slice.Slice method*), 249
items() (*libqtile.layout.spiral.Spiral method*), 250
items() (*libqtile.layout.stack.Stack method*), 252
items() (*libqtile.layout.tile.Tile method*), 254
items() (*libqtile.layout.tree.TreeTab method*), 256
items() (*libqtile.layout.verticaltile.VerticalTile method*), 258
items() (*libqtile.layout.xmonad.MonadTall method*), 238
items() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
items() (*libqtile.layout.xmonad.MonadWide method*), 244
items() (*libqtile.layout.zoomy.Zoomy method*), 259
items() (*libqtile.widget.backlight.Backlight method*), 273
items() (*libqtile.widget.base.Mirror method*), 317
items() (*libqtile.widget.battery.Battery method*), 275
items() (*libqtile.widget.battery.BatteryIcon method*), 276
items() (*libqtile.widget.bluetooth.Bluetooth method*), 277
items() (*libqtile.widget.canto.Canto method*), 280
items() (*libqtile.widget.caps_num_lock_indicator.CapsNumLockIndicator method*), 282
items() (*libqtile.widget.check_updates.CheckUpdates method*), 283
items() (*libqtile.widget.chord.Chord method*), 284
items() (*libqtile.widget.clipboard.Clipboard method*), 285
items() (*libqtile.widget.clock.Clock method*), 286
items() (*libqtile.widget.cmus.Cmus method*), 287
items() (*libqtile.widget.countdown.Countdown method*), 289
items() (*libqtile.widget.cpu.CPU method*), 278
items() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 290
items() (*libqtile.widget.currentlayout.CurrentLayout method*), 291
items() (*libqtile.widget.currentlayout.CurrentLayoutIcon method*), 292
items() (*libqtile.widget.currentscreen.CurrentScreen method*), 293
items() (*libqtile.widget.df.DF method*), 294
items() (*libqtile.widget.do_not_disturb.DoNotDisturb method*), 295
items() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 297
items() (*libqtile.widget.generic_poll_text.GenPollText method*), 298
items() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
items() (*libqtile.widget.gmail_checker.GmailChecker method*), 300
items() (*libqtile.widget.graph.CPUGraph method*), 279
items() (*libqtile.widget.graph.HDDBusyGraph method*), 302
items() (*libqtile.widget.graph.HDDGraph method*), 303
items() (*libqtile.widget.graph.MemoryGraph method*), 316
items() (*libqtile.widget.graph.NetGraph method*), 323
items() (*libqtile.widget.graph.SwapGraph method*), 340
items() (*libqtile.widget.groupbox.AGroupBox method*), 272
items() (*libqtile.widget.groupbox.GroupBox method*), 301
items() (*libqtile.widget.idlerpg.IdleRPG method*), 304
items() (*libqtile.widget.image.Image method*), 306
items() (*libqtile.widget.imapwidget.ImapWidget method*), 307
items() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 308
items() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
items() (*libqtile.widget.khal_calendar.KhalCalendar method*), 310
items() (*libqtile.widget.launchbar.LaunchBar method*), 311
items() (*libqtile.widget.load.Load method*), 313
items() (*libqtile.widget.maildir.Maildir method*), 314
items() (*libqtile.widget.memory.Memory method*), 315
items() (*libqtile.widget.moc.Moc method*), 318
items() (*libqtile.widget.mpd2widget.Mpd2 method*), 319
items() (*libqtile.widget.mpris2widget.Mpris2 method*), 321
items() (*libqtile.widget.net.Net method*), 322
items() (*libqtile.widget.notify.Notify method*), 324
items() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 326
items() (*libqtile.widget.open_weather.OpenWeather method*), 327
items() (*libqtile.widget.pomodoro.Pomodoro method*), 328
items() (*libqtile.widget.prompt.Prompt method*), 330
items() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
items() (*libqtile.widget.quick_exit.QuickExit method*), 332

items() (*libqtile.widget.screensplit.ScreenSplit method*), 334
items() (*libqtile.widget.sensors.ThermalSensor method*), 345
items() (*libqtile.widget.sep.Sep method*), 335
items() (*libqtile.widget.she.She method*), 336
items() (*libqtile.widget.spacer.Spacer method*), 337
items() (*libqtile.widget.statusnotifier.StatusNotifier method*), 338
items() (*libqtile.widget.stock_ticker.StockTicker method*), 339
items() (*libqtile.widget.systray.Systray method*), 342
items() (*libqtile.widget.tasklist.TaskList method*), 343
items() (*libqtile.widget.textbox.TextBox method*), 344
items() (*libqtile.widget.thermal_zone.ThermalZone method*), 347
items() (*libqtile.widget.volume.Volume method*), 348
items() (*libqtile.widget.wallpaper.Wallpaper method*), 349
items() (*libqtile.widget.widgetbox.WidgetBox method*), 350
items() (*libqtile.widget.window_count.WindowCount method*), 352
items() (*libqtile.widget.windowname.WindowName method*), 353
items() (*libqtile.widget.windowtabs.WindowTabs method*), 354
items() (*libqtile.widget.wlan.Wlan method*), 355
items() (*libqtile.widget.wttr.Wttr method*), 356

K

keep_above() (*libqtile.backend.base.window.Window method*), 262
keep_above() (*libqtile.widget.systray.Systray method*), 342
keep_below() (*libqtile.backend.base.window.Window method*), 263
keep_below() (*libqtile.widget.systray.Systray method*), 342
KeyboardKbdd (*class in libqtile.widget.keyboardkbdd*), 307
KeyboardLayout (*class in libqtile.widget.keyboardlayout*), 308
KhalCalendar (*class in libqtile.widget.khal_calendar*), 310
kill() (*libqtile.backend.base.window.Window method*), 263
kill() (*libqtile.widget.systray.Systray method*), 342

L

labelgroup() (*libqtile.core.manager.Qtile method*), 222
LaunchBar (*class in libqtile.widget.launchbar*), 311
layout_change() (*libqtile.hook.subscribe method*), 185
leave_chord() (*libqtile.hook.subscribe method*), 186
left() (*libqtile.layout.bsp.Bsp method*), 229
left() (*libqtile.layout.columns.Columns method*), 232
left() (*libqtile.layout.matrix.Matrix method*), 235
left() (*libqtile.layout.xmonad.MonadTall method*), 238
left() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
left() (*libqtile.layout.xmonad.MonadWide method*), 244
list_widgets() (*libqtile.core.manager.Qtile method*), 222
Load (*class in libqtile.widget.load*), 312
loglevel() (*libqtile.core.manager.Qtile method*), 222
loglevelname() (*libqtile.core.manager.Qtile method*), 222

M

Maildir (*class in libqtile.widget.maildir*), 313
Matrix (*class in libqtile.layout.matrix*), 234
Max (*class in libqtile.layout.max*), 235
maximize() (*libqtile.layout.verticaltile.VerticalTile method*), 258
maximize() (*libqtile.layout.xmonad.MonadTall method*), 238
maximize() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
maximize() (*libqtile.layout.xmonad.MonadWide method*), 244
Memory (*class in libqtile.widget.memory*), 314
MemoryGraph (*class in libqtile.widget.graph*), 315
Mirror (*class in libqtile.widget.base*), 316
Moc (*class in libqtile.widget.moc*), 317
MonadTall (*class in libqtile.layout.xmonad*), 237
MonadThreeCol (*class in libqtile.layout.xmonad*), 239
MonadWide (*class in libqtile.layout.xmonad*), 242
move_down() (*libqtile.backend.base.window.Window method*), 263
move_down() (*libqtile.layout.tree.TreeTab method*), 256
move_down() (*libqtile.widget.systray.Systray method*), 342
move_floating() (*libqtile.backend.base.window.Window method*), 263
move_left() (*libqtile.layout.tree.TreeTab method*), 256
move_right() (*libqtile.layout.tree.TreeTab method*), 256
move_to_bottom() (*libqtile.backend.base.window.Window method*), 263
move_to_bottom() (*libqtile.widget.systray.Systray method*), 342
move_to_slice() (*libqtile.layout.slice.Slice method*), 249
move_to_top() (*libqtile.backend.base.window.Window method*), 263
move_to_top() (*libqtile.widget.systray.Systray method*), 342

move_up() (*libqtile.backend.base.window.Window method*), 263
move_up() (*libqtile.layout.tree.TreeTab method*), 256
move_up() (*libqtile.widget.systray.Systray method*), 342
move_window_to_next_split() (*libqtile.layout.screensplit.ScreenSplit method*), 247
move_window_to_previous_split() (*libqtile.layout.screensplit.ScreenSplit method*), 247
Mpd2 (*class in libqtile.widget.mpd2widget*), 319
Mpris2 (*class in libqtile.widget.mpris2widget*), 320
mute() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
mute() (*libqtile.widget.volume.Volume method*), 348

N

Net (*class in libqtile.widget.net*), 321
net_wm_icon_change() (*libqtile.hook.subscribe method*), 186
NetGraph (*class in libqtile.widget.graph*), 322
next() (*libqtile.layout.bsp.Bsp method*), 229
next() (*libqtile.layout.columns.Columns method*), 232
next() (*libqtile.layout.floating.Floating method*), 234
next() (*libqtile.layout.matrix.Matrix method*), 235
next() (*libqtile.layout.max.Max method*), 236
next() (*libqtile.layout.ratiotile.RatioTile method*), 246
next() (*libqtile.layout.screensplit.ScreenSplit method*), 247
next() (*libqtile.layout.slice.Slice method*), 249
next() (*libqtile.layout.spiral.Spiral method*), 250
next() (*libqtile.layout.stack.Stack method*), 252
next() (*libqtile.layout.tile.Tile method*), 254
next() (*libqtile.layout.tree.TreeTab method*), 256
next() (*libqtile.layout.verticaltile.VerticalTile method*), 258
next() (*libqtile.layout.xmonad.MonadTall method*), 238
next() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
next() (*libqtile.layout.xmonad.MonadWide method*), 244
next() (*libqtile.layout.zoomy.Zoomy method*), 259
next() (*libqtile.widget.mpris2widget.Mpris2 method*), 321
next() (*libqtile.widget.notify.Notify method*), 325
next_group() (*libqtile.config.Screen method*), 358
next_keyboard() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
next_layout() (*libqtile.core.manager.Qtile method*), 222
next_load() (*libqtile.widget.load.Load method*), 313
next_screen() (*libqtile.core.manager.Qtile method*), 222
next_split() (*libqtile.layout.screensplit.ScreenSplit method*), 247
next_urgent() (*libqtile.core.manager.Qtile method*), 222
next_window() (*libqtile.group._Group method*), 267
normalize() (*libqtile.layout.bsp.Bsp method*), 229
normalize() (*libqtile.layout.columns.Columns method*), 232
normalize() (*libqtile.layout.verticaltile.VerticalTile method*), 258
normalize() (*libqtile.layout.xmonad.MonadTall method*), 238
normalize() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
normalize() (*libqtile.layout.xmonad.MonadWide method*), 244
Notify (*class in libqtile.widget.notify*), 323
NvidiaSensors (*class in libqtile.widget.nvidia_sensors*), 325

O

open() (*libqtile.widget.widgetbox.WidgetBox method*), 351
OpenWeather (*class in libqtile.widget.open_weather*), 326

P

pause() (*libqtile.core.manager.Qtile method*), 222
place() (*libqtile.backend.base.window.Window method*), 263
place() (*libqtile.widget.systray.Systray method*), 342
play_pause() (*libqtile.widget.mpris2widget.Mpris2 method*), 321
Pomodoro (*class in libqtile.widget.pomodoro*), 327
prev() (*libqtile.widget.notify.Notify method*), 325
prev_group() (*libqtile.config.Screen method*), 358
prev_layout() (*libqtile.core.manager.Qtile method*), 222
prev_screen() (*libqtile.core.manager.Qtile method*), 222
prev_window() (*libqtile.group._Group method*), 267
previous() (*libqtile.layout.bsp.Bsp method*), 229
previous() (*libqtile.layout.columns.Columns method*), 232
previous() (*libqtile.layout.floating.Floating method*), 234
previous() (*libqtile.layout.matrix.Matrix method*), 235
previous() (*libqtile.layout.max.Max method*), 236
previous() (*libqtile.layout.ratiotile.RatioTile method*), 246
previous() (*libqtile.layout.screensplit.ScreenSplit method*), 248
previous() (*libqtile.layout.slice.Slice method*), 249
previous() (*libqtile.layout.spiral.Spiral method*), 250

previous() (*libqtile.layout.stack.Stack method*), 252
previous() (*libqtile.layout.tile.Tile method*), 254
previous() (*libqtile.layout.tree.TreeTab method*), 256
previous() (*libqtile.layout.verticaltile.VerticalTile method*), 258
previous() (*libqtile.layout.xmonad.MonadTall method*), 238
previous() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
previous() (*libqtile.layout.xmonad.MonadWide method*), 244
previous() (*libqtile.layout.zoomy.Zoomy method*), 259
previous() (*libqtile.widget.mpris2widget.Mpris2 method*), 321
previous_split() (*libqtile.layout.screensplit.ScreenSplit method*), 248
Prompt (*class in libqtile.widget.prompt*), 329
PulseVolume (*class in libqtile.widget.pulse_volume*), 330

Q

Qtile (*class in libqtile.core.manager*), 218
qtile_info() (*libqtile.core.manager.Qtile method*), 223
qtilecmd() (*libqtile.core.manager.Qtile method*), 223
query_tree() (*libqtile.backend.wayland.core.Core method*), 362
QuickExit (*class in libqtile.widget.quick_exit*), 332

R

RatioTile (*class in libqtile.layout.ratiotile*), 245
reconfigure_screens() (*libqtile.core.manager.Qtile method*), 223
reload_config() (*libqtile.core.manager.Qtile method*), 223
remove_rule() (*libqtile.core.manager.Qtile method*), 223
reset() (*libqtile.layout.spiral.Spiral method*), 250
reset() (*libqtile.layout.tile.Tile method*), 254
reset() (*libqtile.layout.xmonad.MonadTall method*), 238
reset() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
reset() (*libqtile.layout.xmonad.MonadWide method*), 244
resize() (*libqtile.config.Screen method*), 358
resize_floating() (*libqtile.backend.base.window.Window method*), 263
restart() (*libqtile.core.manager.Qtile method*), 223
restart() (*libqtile.hook.subscribe method*), 186
resume() (*libqtile.hook.subscribe method*), 186
right() (*libqtile.layout.bsp.Bsp method*), 229
right() (*libqtile.layout.columns.Columns method*), 232
right() (*libqtile.layout.matrix.Matrix method*), 235
right() (*libqtile.layout.xmonad.MonadTall method*), 238
right() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
right() (*libqtile.layout.xmonad.MonadWide method*), 244
rotate() (*libqtile.layout.stack.Stack method*), 252
run_app() (*libqtile.widget.pulse_volume.PulseVolume method*), 331
run_app() (*libqtile.widget.volume.Volume method*), 348
run_extension() (*libqtile.core.manager.Qtile method*), 223

S

Screen (*class in libqtile.config*), 357
screen_change() (*libqtile.hook.subscribe method*), 187
screens_reconfigured() (*libqtile.hook.subscribe method*), 187
ScreenSplit (*class in libqtile.layout.screensplit*), 246
ScreenSplit (*class in libqtile.widget.screensplit*), 333
scroll_down() (*libqtile.widget.bluetooth.Bluetooth method*), 277
scroll_up() (*libqtile.widget.bluetooth.Bluetooth method*), 277
section_down() (*libqtile.layout.tree.TreeTab method*), 256
section_up() (*libqtile.layout.tree.TreeTab method*), 256
selection_change() (*libqtile.hook.subscribe method*), 187
selection_notify() (*libqtile.hook.subscribe method*), 188
Sep (*class in libqtile.widget.sep*), 334
set_font() (*libqtile.widget.backlight.Backlight method*), 274
set_font() (*libqtile.widget.battery.Battery method*), 275
set_font() (*libqtile.widget.bluetooth.Bluetooth method*), 277
set_font() (*libqtile.widget.canto.Canto method*), 281
set_font() (*libqtile.widget.caps_num_lock_indicator.CapsNumLockIndic method*), 282
set_font() (*libqtile.widget.check_updates.CheckUpdates method*), 283
set_font() (*libqtile.widget.chord.Chord method*), 284
set_font() (*libqtile.widget.clipboard.Clipboard method*), 285
set_font() (*libqtile.widget.clock.Clock method*), 286
set_font() (*libqtile.widget.cmus.Cmus method*), 288
set_font() (*libqtile.widget.countdown.Countdown method*), 289
set_font() (*libqtile.widget.cpu.CPU method*), 278
set_font() (*libqtile.widget.crypto_ticker.CryptoTicker method*), 290

set_font() (*libqtile.widget.currentlayout.CurrentLayout method*), 291
set_font() (*libqtile.widget.currentlayout.CurrentLayoutIcon method*), 292
set_font() (*libqtile.widget.currentscreen.CurrentScreen method*), 293
set_font() (*libqtile.widget.df.DF method*), 295
set_font() (*libqtile.widget.do_not_disturb.DoNotDisturb method*), 296
set_font() (*libqtile.widget.generic_poll_text.GenPollCommand method*), 297
set_font() (*libqtile.widget.generic_poll_text.GenPollText method*), 298
set_font() (*libqtile.widget.generic_poll_text.GenPollUrl method*), 299
set_font() (*libqtile.widget.gmail_checker.GmailChecker method*), 300
set_font() (*libqtile.widget.groupbox.AGroupBox method*), 272
set_font() (*libqtile.widget.groupbox.GroupBox method*), 301
set_font() (*libqtile.widget.idlerpg.IdleRPG method*), 305
set_font() (*libqtile.widget.imapwidget.ImapWidget method*), 307
set_font() (*libqtile.widget.keyboardkbdd.KeyboardKbdd method*), 308
set_font() (*libqtile.widget.keyboardlayout.KeyboardLayout method*), 309
set_font() (*libqtile.widget.khal_calendar.KhalCalendar method*), 311
set_font() (*libqtile.widget.load.Load method*), 313
set_font() (*libqtile.widget.maildir.Maildir method*), 314
set_font() (*libqtile.widget.memory.Memory method*), 315
set_font() (*libqtile.widget.moc.Moc method*), 318
set_font() (*libqtile.widget.mpd2widget.Mpd2 method*), 320
set_font() (*libqtile.widget.mpris2widget.Mpris2 method*), 321
set_font() (*libqtile.widget.net.Net method*), 322
set_font() (*libqtile.widget.notify.Notify method*), 325
set_font() (*libqtile.widget.nvidia_sensors.NvidiaSensors method*), 326
set_font() (*libqtile.widget.open_weather.OpenWeather method*), 327
set_font() (*libqtile.widget.pomodoro.Pomodoro method*), 328
set_font() (*libqtile.widget.prompt.Prompt method*), 330
set_font() (*libqtile.widget.pulse_volume.PulseVolume method*), 332
set_font() (*libqtile.widget.quick_exit.QuickExit method*), 333
set_font() (*libqtile.widget.screensplit.ScreenSplit method*), 334
set_font() (*libqtile.widget.sensors.ThermalSensor method*), 346
set_font() (*libqtile.widget.she.She method*), 336
set_font() (*libqtile.widget.stock_ticker.StockTicker method*), 339
set_font() (*libqtile.widget.textbox.TextBox method*), 345
set_font() (*libqtile.widget.thermal_zone.ThermalZone method*), 347
set_font() (*libqtile.widget.volume.Volume method*), 348
set_font() (*libqtile.widget.wallpaper.Wallpaper method*), 349
set_font() (*libqtile.widget.widgetbox.WidgetBox method*), 351
set_font() (*libqtile.widget.window_count.WindowCount method*), 352
set_font() (*libqtile.widget.windowname.WindowName method*), 353
set_font() (*libqtile.widget.windowtabs.WindowTabs method*), 354
set_font() (*libqtile.widget.wlan.Wlan method*), 355
set_font() (*libqtile.widget.wttr.Wttr method*), 356
set_keymap() (*libqtile.backend.wayland.core.Core method*), 362
set_label() (*libqtile.group._Group method*), 267
set_master_ratio() (*libqtile.layout.spiral.Spiral method*), 250
set_opacity() (*libqtile.backend.base.window.Window method*), 263
set_position() (*libqtile.backend.base.window.Window method*), 263
set_position_floating() (*libqtile.backend.base.window.Window method*), 263
set_ratio() (*libqtile.layout.spiral.Spiral method*), 250
set_ratio() (*libqtile.layout.xmonad.MonadTall method*), 238
set_ratio() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
set_ratio() (*libqtile.layout.xmonad.MonadWide method*), 244
set_size_floating() (*libqtile.backend.base.window.Window method*), 263
set_wallpaper() (*libqtile.config.Screen method*), 358
setgroup() (*libqtile.hook.subscribe method*), 188
setlayout() (*libqtile.group._Group method*), 267
She (*class in libqtile.widget.she*), 335
shrink() (*libqtile.layout.verticaltile.VerticalTile method*), 258

shrink() (*libqtile.layout.xmonad.MonadTall method*), 239
shrink() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
shrink() (*libqtile.layout.xmonad.MonadWide method*), 244
shrink_main() (*libqtile.layout.spiral.Spiral method*), 250
shrink_main() (*libqtile.layout.xmonad.MonadTall method*), 239
shrink_main() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
shrink_main() (*libqtile.layout.xmonad.MonadWide method*), 244
shuffle_down() (*libqtile.layout.bsp.Bsp method*), 229
shuffle_down() (*libqtile.layout.columns.Columns method*), 232
shuffle_down() (*libqtile.layout.ratiotile.RatioTile method*), 246
shuffle_down() (*libqtile.layout.spiral.Spiral method*), 250
shuffle_down() (*libqtile.layout.stack.Stack method*), 252
shuffle_down() (*libqtile.layout.tile.Tile method*), 254
shuffle_down() (*libqtile.layout.verticaltile.VerticalTile method*), 258
shuffle_down() (*libqtile.layout.xmonad.MonadTall method*), 239
shuffle_down() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
shuffle_down() (*libqtile.layout.xmonad.MonadWide method*), 244
shuffle_left() (*libqtile.layout.bsp.Bsp method*), 229
shuffle_left() (*libqtile.layout.columns.Columns method*), 232
shuffle_right() (*libqtile.layout.bsp.Bsp method*), 229
shuffle_right() (*libqtile.layout.columns.Columns method*), 232
shuffle_up() (*libqtile.layout.bsp.Bsp method*), 230
shuffle_up() (*libqtile.layout.columns.Columns method*), 232
shuffle_up() (*libqtile.layout.ratiotile.RatioTile method*), 246
shuffle_up() (*libqtile.layout.spiral.Spiral method*), 250
shuffle_up() (*libqtile.layout.stack.Stack method*), 252
shuffle_up() (*libqtile.layout.tile.Tile method*), 254
shuffle_up() (*libqtile.layout.verticaltile.VerticalTile method*), 258
shuffle_up() (*libqtile.layout.xmonad.MonadTall method*), 239
shuffle_up() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
shuffle_up() (*libqtile.layout.xmonad.MonadWide method*), 245
shutdown() (*libqtile.core.manager.Qtile method*), 223
shutdown() (*libqtile.hook.subscribe method*), 189
simple_key_binder() (*in module libqtile.dgroups*), 48
simulate_keypress() (*libqtile.core.manager.Qtile method*), 223
Slice (*class in libqtile.layout.slice*), 248
sort_windows() (*libqtile.layout.tree.TreeTab method*), 256
Spacer (*class in libqtile.widget.spacer*), 336
spawn() (*libqtile.core.manager.Qtile method*), 224
spawncmd() (*libqtile.core.manager.Qtile method*), 224
Spiral (*class in libqtile.layout.spiral*), 249
Stack (*class in libqtile.layout.stack*), 251
startup() (*libqtile.hook.subscribe method*), 189
startup_complete() (*libqtile.hook.subscribe method*), 189
startup_once() (*libqtile.hook.subscribe method*), 190
static() (*libqtile.backend.base.window.Window method*), 263
status() (*libqtile.core.manager.Qtile method*), 224
StatusNotifier (*class in libqtile.widget.statusnotifier*), 337
StockTicker (*class in libqtile.widget.stock_ticker*), 338
stop() (*libqtile.widget.mpris2widget.Mpris2 method*), 321
suspend() (*libqtile.hook.subscribe method*), 190
swap() (*libqtile.layout.xmonad.MonadTall method*), 239
swap() (*libqtile.layout.xmonad.MonadThreeCol method*), 241
swap() (*libqtile.layout.xmonad.MonadWide method*), 245
swap_column_left() (*libqtile.layout.columns.Columns method*), 232
swap_column_right() (*libqtile.layout.columns.Columns method*), 232
swap_left() (*libqtile.layout.xmonad.MonadTall method*), 239
swap_left() (*libqtile.layout.xmonad.MonadThreeCol method*), 242
swap_left() (*libqtile.layout.xmonad.MonadWide method*), 245
swap_main() (*libqtile.layout.xmonad.MonadTall method*), 239
swap_main() (*libqtile.layout.xmonad.MonadThreeCol method*), 242
swap_main() (*libqtile.layout.xmonad.MonadWide method*), 245
swap_right() (*libqtile.layout.xmonad.MonadTall method*), 239
swap_right() (*libqtile.layout.xmonad.MonadThreeCol method*), 242
swap_right() (*libqtile.layout.xmonad.MonadWide*

method), 245
swap_window_order() (*libqtile.group._Group method*), 267
SwapGraph (*class in libqtile.widget.graph*), 339
switch_groups() (*libqtile.core.manager.Qtile method*), 224
switch_groups() (*libqtile.group._Group method*), 267
switch_window() (*libqtile.core.manager.Qtile method*), 224
switchgroup() (*libqtile.core.manager.Qtile method*), 224
sync() (*libqtile.core.manager.Qtile method*), 225
Systray (*class in libqtile.widget.systray*), 340

T

TaskList (*class in libqtile.widget.tasklist*), 343
TextBox (*class in libqtile.widget.textbox*), 344
ThermalSensor (*class in libqtile.widget.sensors*), 345
ThermalZone (*class in libqtile.widget.thermal_zone*), 346
Tile (*class in libqtile.layout.tile*), 253
to_layout_index() (*libqtile.core.manager.Qtile method*), 225
to_screen() (*libqtile.core.manager.Qtile method*), 225
toggle() (*libqtile.widget.notify.Notify method*), 325
toggle() (*libqtile.widget.widgetbox.WidgetBox method*), 351
toggle_active() (*libqtile.widget.pomodoro.Pomodoro method*), 328
toggle_break() (*libqtile.widget.pomodoro.Pomodoro method*), 328
toggle_floating() (*libqtile.backend.base.window.Window method*), 264
toggle_fullscreen() (*libqtile.backend.base.window.Window method*), 264
toggle_group() (*libqtile.config.Screen method*), 359
toggle_maximize() (*libqtile.backend.base.window.Window method*), 264
toggle_minimize() (*libqtile.backend.base.window.Window method*), 264
toggle_split() (*libqtile.layout.bsp.Bsp method*), 230
toggle_split() (*libqtile.layout.columns.Columns method*), 232
toggle_split() (*libqtile.layout.stack.Stack method*), 252
togroup() (*libqtile.backend.base.window.Window method*), 264
togroup() (*libqtile.core.manager.Qtile method*), 225
toscreen() (*libqtile.backend.base.window.Window method*), 264
toscreen() (*libqtile.group._Group method*), 267
tracemalloc_dump() (*libqtile.core.manager.Qtile method*), 225
tracemalloc_toggle() (*libqtile.core.manager.Qtile method*), 225
TreeTab (*class in libqtile.layout.tree*), 254
trigger() (*libqtile.widget.quick_exit.QuickExit method*), 333

U

ungrab_all_chords() (*libqtile.core.manager.Qtile method*), 225
ungrab_chord() (*libqtile.core.manager.Qtile method*), 225
unhide_cursor() (*libqtile.backend.wayland.core.Core method*), 362
unminimize_all() (*libqtile.group._Group method*), 268
up() (*libqtile.layout.bsp.Bsp method*), 230
up() (*libqtile.layout.columns.Columns method*), 232
up() (*libqtile.layout.matrix.Matrix method*), 235
up() (*libqtile.layout.max.Max method*), 236
up() (*libqtile.layout.stack.Stack method*), 252
up_opacity() (*libqtile.backend.base.window.Window method*), 265
update() (*libqtile.widget.image.Image method*), 306
update() (*libqtile.widget.screensplit.ScreenSplit method*), 334
update() (*libqtile.widget.textbox.TextBox method*), 345
update_timezone() (*libqtile.widget.clock.Clock method*), 286
use_system_timezone() (*libqtile.widget.clock.Clock method*), 286
user() (*libqtile.hook.subscribe method*), 191

V

validate_config() (*libqtile.core.manager.Qtile method*), 225
VerticalTile (*class in libqtile.layout.verticaltile*), 257
Volume (*class in libqtile.widget.volume*), 347

W

Wallpaper (*class in libqtile.widget.wallpaper*), 348
warning() (*libqtile.core.manager.Qtile method*), 225
WidgetBox (*class in libqtile.widget.widgetbox*), 350
Window (*class in libqtile.backend.base.window*), 260
WindowCount (*class in libqtile.widget.window_count*), 351
WindowName (*class in libqtile.widget.windowname*), 352
windows() (*libqtile.core.manager.Qtile method*), 226
WindowTabs (*class in libqtile.widget.windowtabs*), 353
Wlan (*class in libqtile.widget.wlan*), 354
Wttr (*class in libqtile.widget.wttr*), 355

Z

Zoomy (*class in libqtile.layout.zoomy*), 258