## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Getting started</td>
<td>1</td>
</tr>
<tr>
<td>2 Reference</td>
<td>41</td>
</tr>
<tr>
<td>3 Advanced scripting</td>
<td>117</td>
</tr>
<tr>
<td>4 Getting involved</td>
<td>135</td>
</tr>
<tr>
<td>5 Miscellaneous</td>
<td>143</td>
</tr>
<tr>
<td>6 Tips &amp; Tricks</td>
<td>147</td>
</tr>
<tr>
<td>Index</td>
<td>159</td>
</tr>
</tbody>
</table>
1.1 Installing Qtile

1.1.1 Distro Guides

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

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.

Installing on Fedora

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

```
dnf -y install qtile
```

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.

Installing on Debian or Ubuntu

Note: As of Ubuntu 20.04 (Focal Fossa), the package has been outdated and removed from the Ubuntu's official package list. Users are advised to follow the instructions of Installing From Source.

On other recent Ubuntu (17.04 or greater) and Debian unstable versions, there are Qtile packages available via:

```
sudo apt-get install qtile
```

On older versions of Ubuntu (15.10 to 16.10) and Debian 9, the dependencies are available via:

```
sudo apt-get install python3-xcffib python3-cairocffi
```

Installing on Slackware

Qtile is available on the SlackBuilds.org as:

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qtile</td>
<td>stable branch (release)</td>
</tr>
</tbody>
</table>

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.

Installing on FreeBSD

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

```
pkg install qtile
```

### 1.1.2 Installing From Source

**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.

**Core Dependencies**

Here are Qtile’s core runtime dependencies and where available the package name that provides 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.

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Ubuntu Package</th>
<th>Needed for</th>
</tr>
</thead>
<tbody>
<tr>
<td>CFFI</td>
<td>python3-cffi</td>
<td>Both backends, bars and popups</td>
</tr>
<tr>
<td>X server</td>
<td>xserver-xorg</td>
<td>X11 backend</td>
</tr>
<tr>
<td>xcffib</td>
<td>python3-xcffib</td>
<td>X11 backend</td>
</tr>
<tr>
<td>wlroots</td>
<td>libwlroots-dev</td>
<td>Wayland backend (see below)</td>
</tr>
<tr>
<td>pywlroots</td>
<td>--</td>
<td>Wayland backend</td>
</tr>
<tr>
<td>pywayland</td>
<td>--</td>
<td>Wayland backend</td>
</tr>
<tr>
<td>python-xkbcommon</td>
<td>--</td>
<td>Wayland backend</td>
</tr>
<tr>
<td>cairocffi</td>
<td>python3-cairocffi</td>
<td>Drawing on bars and popups (see below)</td>
</tr>
<tr>
<td>libpangocairo</td>
<td>libpangocairo-1.0-0</td>
<td>Writing on bars and popups</td>
</tr>
<tr>
<td>dbus-next</td>
<td>--</td>
<td>Sending notifications with dbus (optional)</td>
</tr>
</tbody>
</table>
Qtile uses cairocffi for drawing on status bars and popup windows. Under X11, cairocffi requires XCB support via xcfib, which you should be sure to have installed before installing cairocffi, otherwise the needed cairo-xcb bindings will not be built. Once you've got the dependencies installed, you can use the latest version on PyPI:

```bash
pip install --no-cache-dir cairocffi
```

Qtile

With the dependencies in place, you can now install qtile:

```bash
git clone git://github.com/qtile/qtile.git
cd qtile
pip install .
```

Stable versions of Qtile can be installed from PyPI:

```bash
pip install qtile
```

As long as the necessary libraries are in place, this can be done at any point, however, it is recommended that you first install xcfib to ensure the cairo-xcb bindings are built (X11 only) (see above).

1.1.3 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. This means we can only support the latest release. Be aware that some distributions package outdated versions of wlroots. More up-to-date distributions such as Arch Linux may also package pywayland, pywlroots and python-xkbcommon.

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:

```bash
qtile start -b wayland
```

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

1.2 Configuration

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.
1.2.1 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`
- 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.

1.2.2 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.

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.
Mouse Bindings

By default, holding your mod key and clicking (and holding) a window will allow you to drag it around as a floating window.

1.2.3 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!)

Lazy objects

The lazy.lazy object is a special helper object to specify a command for later execution. This object acts like the root of the object graph, which means that we can specify a key binding command with the same syntax used to call the command through a script or through qtile shell.

Example

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

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

Lazy functions

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

Some examples are given below.
General functions

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy.spawn(&quot;application&quot;)</td>
<td>Run the application</td>
</tr>
<tr>
<td>lazy.spawncmd()</td>
<td>Open command prompt on the bar. See prompt widget.</td>
</tr>
<tr>
<td>lazy.reload_config()</td>
<td>Reload the config.</td>
</tr>
<tr>
<td>lazy.restart()</td>
<td>Restart Qtile. In X11, it won’t close your windows.</td>
</tr>
<tr>
<td>lazy.shutdown()</td>
<td>Close the whole Qtile</td>
</tr>
</tbody>
</table>

Group functions

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy.next_layout()</td>
<td>Use next layout on the actual group</td>
</tr>
<tr>
<td>lazy.prev_layout()</td>
<td>Use previous layout on the actual group</td>
</tr>
<tr>
<td>lazy.screen.next_group()</td>
<td>Move to the group on the right</td>
</tr>
<tr>
<td>lazy.screen.prev_group()</td>
<td>Move to the group on the left</td>
</tr>
<tr>
<td>lazy.screen.toggle_group()</td>
<td>Move to the last visited group</td>
</tr>
<tr>
<td>lazy.group.next_window()</td>
<td>Switch window focus to next window in group</td>
</tr>
<tr>
<td>lazy.group.prev_window()</td>
<td>Switch window focus to previous window in group</td>
</tr>
<tr>
<td>lazy.group[&quot;group_name&quot;].toscreen()</td>
<td>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.</td>
</tr>
<tr>
<td>lazy.layout.increase_ratio()</td>
<td>Increase the space for master window at the expense of slave windows</td>
</tr>
<tr>
<td>lazy.layout.decrease_ratio()</td>
<td>Decrease the space for master window in the advantage of slave windows</td>
</tr>
</tbody>
</table>
Window functions

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy.window.kill()</td>
<td>Close the focused window</td>
</tr>
<tr>
<td>lazy.layout.next()</td>
<td>Switch window focus to other pane(s) of stack</td>
</tr>
<tr>
<td>lazy.window.togroup(&quot;group_name&quot;)</td>
<td>Move focused window to the group called group_name</td>
</tr>
<tr>
<td>lazy.window.toggle_floating()</td>
<td>Put the focused window to/from floating mode</td>
</tr>
<tr>
<td>lazy.window.toggle_fullscreen()</td>
<td>Put the focused window to/from fullscreen mode</td>
</tr>
</tbody>
</table>

ScratchPad DropDown functions

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy.group[&quot;group_name&quot;].dropdown_toggle(&quot;name&quot;)</td>
<td>Toggles the visibility of the specified DropDown window. On first use, the configured process is spawned.</td>
</tr>
<tr>
<td>lazy.group[&quot;group_name&quot;].hide_all()</td>
<td>Hides all DropDown windows.</td>
</tr>
<tr>
<td>lazy.group[&quot;group_name&quot;].dropdown_reconfigure(&quot;name&quot;, **configuration)</td>
<td>Update the configuration of the named DropDown.</td>
</tr>
</tbody>
</table>

User-defined functions

<table>
<thead>
<tr>
<th>function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lazy.function(func, *args, **kwargs)</td>
<td>Calls func(qtile, *args, **kwargs). NB. the qtile object is automatically passed as the first argument.</td>
</tr>
</tbody>
</table>

Examples

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

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

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

(continues on next page)
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.

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

```python
from libqtile.config import Key
from libqtile.command 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)  
    )  
]```
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 `DGroup` objects.

DGroup 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

```python
from libqtile.config import Group, Match

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

# 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

```python
class libqtile.config.Group(name: str, matches: Optional[List[libqtile.config.Match]] = None,
exclusive=False, spawn: Optional[Union[str, List[str]]] = None, layout:
Optional[str] = None, layouts: Optional[List] = None, persist=True, init=True,
layout_opts=None, screen_affinity=None, position=9223372036854775807,
label: Optional[str] = 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**: string the name of this group
- **matches**: default `None` list of `Match` objects whose windows will be assigned to this group
- **exclusive**: boolean when other apps are started in this group, should we allow them here or not?
- **spawn**: string or list of strings this will be `exec()` d when the group is created, you can pass either a program name or a list of programs to `exec()`
- **layout**: string the name of default layout for this group (e.g. 'max' or 'stack'). This is the name specified for a particular layout in config.py or if not defined it defaults in general the class name in all lower case.
- **layouts**: list the group layouts list overriding global layouts. Use this to define a separate list of layouts for this particular group.
**persist**: boolean  
should this group stay alive with no member windows?

**init**: boolean  
is this group alive when qtile starts?

**position int**  
group position

**label**: string  
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.

```python
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**

```python
class libqtile.config.Match(title=None, wm_class=None, role=None, wm_type=None, wm_instance_class=None, net_wm_pid=None, func: Optional[Callable[[Union[libqtile.backend.base.Window, libqtile.backend.base.Internal, libqtile.backend.base.Static]], bool]] = None, wid=None)
```

Match for dynamic groups or auto-floating windows.

It can match by title, wm_class, role, wm_type, wm_instance_class or net_wm_pid.

**Match** supports both regular expression objects (i.e. the result of `re.compile()`) or strings (match as an "include"-match). If a window matches all specified values, it is considered a match.

**Parameters**

- **title**: matches against the WM_NAME atom (X11) or title (Wayland)
- **wm_class**: matches against the second string in WM_CLASS atom (X11) or app ID (Wayland)
- **role**: matches against the WM_ROLE atom (X11 only)
- **wm_type**: matches against the WM_TYPE atom (X11 only)
- **wm_instance_class**: matches against the first string in WM_CLASS atom (X11) or app ID (Wayland)
- **net_wm_pid**: matches against the _NET_WM_PID atom (X11) or PID (Wayland) - (only int allowed for this rule)
- **func**: delegate the match to the given function, which receives the tested client as argument and must return True if it matches, False otherwise

**Rule**

```python
class libqtile.config.Rule(match=None, group=None, float=False, intrusive=False, break_on_match=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**  
  auto float this window?

- **intrusive**  
  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 that 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

```python
from libqtile.config import Group, ScratchPad, DropDown, Key
from libqtile.command 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 visibility of above defined DropDown named "term"
    Key([], 'F11', lazy.group["scratchpad"].dropdown_toggle("term")),
    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 config.Match object to the corresponding DropDown. See below.

Reference

ScratchPad

class libqtile.config.ScratchPad(name, dropdowns=None, position=922372036854775807, label="", single=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**: string the name of this group
- **dropdowns**: default `'None` list of DropDown objects
- **position**: int group position
- **label**: string The display name of the ScratchPad group. Defaults to the empty string such that the group is hidden in `GroupList` widget.
- **single**: [Boolean] Only one of the window among the specified dropdowns will be visible at a time.

**DropDown**

class `libqtile.config.DropDown(name, cmd, **config)`
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.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>height</strong></td>
<td>0.35</td>
<td>Height of window as fraction of current screen.</td>
</tr>
<tr>
<td><strong>match</strong></td>
<td>None</td>
<td>Use a config.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.</td>
</tr>
<tr>
<td><strong>on_focus_lost_hide</strong></td>
<td>True</td>
<td>Shall the window be hidden if focus is lost? If so, the DropDown is hidden if window focus or the group is changed.</td>
</tr>
<tr>
<td><strong>opacity</strong></td>
<td>0.9</td>
<td>Opacity of window as fraction. Zero is opaque.</td>
</tr>
<tr>
<td><strong>warp_pointer</strong></td>
<td>True</td>
<td>Shall pointer warp to center of window on activation? This has only effect if any of the on_focus_lost_xxx configurations is True</td>
</tr>
<tr>
<td><strong>width</strong></td>
<td>0.8</td>
<td>Width of window as fraction of current screen width</td>
</tr>
<tr>
<td><strong>x</strong></td>
<td>0.1</td>
<td>X position of window as fraction of current screen width. 0 is the left most position.</td>
</tr>
<tr>
<td><strong>y</strong></td>
<td>0.0</td>
<td>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 &lt; 1 and an appropriate height.</td>
</tr>
</tbody>
</table>

**Keys**

The `keys` variable defines Qtile's key bindings. Individual key bindings are defined with `libqtile.config.Key` as demonstrated in the following example. Note that you may specify more than one callback functions.

```python
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".
]```

(continues on next page)
The above may also be written more concisely with the help of the `libqtile.config.EzKey` helper class. The following example is functionally equivalent to the above:

```python
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:

```python
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:

```python
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)
    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. In Qtile, these sequences are known as chords and are defined with `libqtile.config.KeyChord`. Chords are added to the `keys` section of the config file.

```python
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 focused window. If <escape> has not been pressed, the next press of x will launch xterm.

Modes

Chords can optionally specify a "mode". When this is done, the mode will remain active until the user presses <escape>. This can be useful for configuring a subset of commands for a particular situations (i.e. similar to vim modes).

```python
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="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:** If using modes, users may also wish to use the Chord widget (`libqtile.widget.chord.Chord`) as this will display the name of the currently active mode on the bar.
Chains

Chords can also be chained to make even longer sequences.

```python
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:

```python
from libqtile.config import Key, KeyChord

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

After pressing Mod+z y c x, 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:

```python
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())`.

### Reference

#### Key

```python
class libqtile.config.Key(modifiers: List[str], key: str, *commands, desc: str = "")
```

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**: A list of lazy command objects generated with the lazy.lazy helper. If multiple Call objects are specified, they are run in sequence.
- **desc**: description to be added to the key binding

#### KeyChord

```python
class libqtile.config.KeyChord(modifiers: List[str], key: str, submappings: List[Union[libqtile.config.Key, libqtile.config.KeyChord]], mode: str = "")
```

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: A string with vim like mode name. If it's set, the chord mode will not be left after a keystroke (except for Esc which always leaves the current chord/mode).

EzConfig

class libqtile.config.EzConfig
    Helper class for defining key and button bindings in an emacs-like format. Inspired by Xmonad's XMonad.Util.EZConfig.

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

```python
from libqtile import layout
layouts = [
    layout.Max(),
    layout.Stack(stacks=2)
]
```

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.

Example

```python
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, **kwargs)
    Defines binding of a mouse click

Drag

class libqtile.config.Drag(*args, start=False, **kwargs)
    Defines binding of a mouse to some dragging action

    On each motion event command is executed with two extra parameters added x and y offset from previous move.

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

screens = [
    Screen(
        bottom=bar.Bar([widget.GroupBox(),
                        widget.WindowName()], 30),
    ),
    Screen(
        bottom=bar.Bar([widget.GroupBox(),
                        widget.WindowName()], 30),
    )]
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 background (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](https://wiki.archlinux.org/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:

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

# screens look like this
# 600 300
# |-------------|-----|
# | 480| |580
# | A | B |
# |----------|--| |
# | 400|--|-----|
```
<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td></td>
</tr>
<tr>
<td>400</td>
<td></td>
</tr>
</tbody>
</table>

# Notice there is a hole in the middle
# also D goes down below the others

```python
fake_screens = [
    Screen(
        bottom=bar.Bar(
            widgets=[
                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(
            widgets=[
                widget.GroupBox(),
                widget.WindowName(),
                widget.Clock()
            ],
            30,
        ),
        x=600,
        y=0,
        width=300,
        height=580
    ),
    Screen(
        top=bar.Bar(
            widgets=[
                widget.GroupBox(),
                widget.WindowName(),
                widget.Clock()
            ],
            30,
        ),
        x=0,
    )
]  
```

(continues on next page)
y=480,  
width=500,  
height=400  
),  
Screen(  

top=bar.Bar(  
[

    widget.GroupBox(),  
    widget.WindowName(),  
    widget.Clock()  
],  
30,  
),  
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


A physical screen, and its associated paraphernalia.

Define a screen with a given set of Bars of a specific geometry. Note that bar.Bar objects can only be placed at the top or the bottom of the screen (bar.Gap objects can be placed anywhere). 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.
Bar

class libqtile.bar.Bar(widgets, size, **config)
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.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>'#000000'</td>
<td>Background colour.</td>
</tr>
<tr>
<td>border_color</td>
<td>'#000000'</td>
<td>Border colour as str or list of str [N E S W]</td>
</tr>
<tr>
<td>border_width</td>
<td>0</td>
<td>Width of border as int or list of ints [N E S W]</td>
</tr>
<tr>
<td>margin</td>
<td>0</td>
<td>Space around bar as int or list of ints [N E S W].</td>
</tr>
<tr>
<td>opacity</td>
<td>1</td>
<td>Bar window opacity.</td>
</tr>
</tbody>
</table>

Gap

class libqtile.bar.Gap(size)
A gap placed along one of the edges of the screen

If a gap has been defined, Qtile will avoid covering it with windows. The most probable reason for configuring a gap is to make space for a third-party bar or other static window.

Parameters

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

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:

```python
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` this file so that it can be executed.

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

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

```python
import os
import subprocess
from libqtile import hook

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

```python
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:

```python
@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:
<table>
<thead>
<tr>
<th>variable</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto_fullscreen</td>
<td>True</td>
<td>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.</td>
</tr>
<tr>
<td>bring_froScrWink</td>
<td>Only</td>
<td>When clicked, should the window be brought to the front or not. If this is set to &quot;floating_only&quot;, only floating windows will get affected (This sets the X Stack Mode to Above.)</td>
</tr>
<tr>
<td>cursor_waPllse</td>
<td>True</td>
<td>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.</td>
</tr>
<tr>
<td>dgroups_key_binder</td>
<td>None</td>
<td>A function which generates group binding hotkeys. It takes a single argument, the DGroups object, and can be used 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.</td>
</tr>
<tr>
<td>dgroups_app_rules</td>
<td>[]</td>
<td>A list of Rule objects which can send windows to various groups based on matching criteria.</td>
</tr>
<tr>
<td>floating_layout</td>
<td>Floating([Floating(.])]</td>
<td>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.</td>
</tr>
<tr>
<td>focus_on_window_activation</td>
<td>'smart'</td>
<td>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</td>
</tr>
<tr>
<td>follow_mouse_focus</td>
<td>True</td>
<td>Controls whether or not focus follows the mouse around as it moves across windows in a layout.</td>
</tr>
<tr>
<td>widget_defaults</td>
<td>dict(font=’sans’, fontsize=12, padding=3)</td>
<td>Default settings for bar widgets. Note: if the font file associated with the font selected here is modified while Qtile is running, Qtile may segfault (for details see issue #2656).</td>
</tr>
<tr>
<td>reconfigure_screens</td>
<td>True</td>
<td>Controls whether or not to automatically reconfigure screens when there are changes in randr output configuration.</td>
</tr>
<tr>
<td>wmname</td>
<td>'LG3D'</td>
<td>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.</td>
</tr>
<tr>
<td>auto_minimize</td>
<td>True</td>
<td>If things like steam games want to auto-minimize themselves when losing focus, should we respect this or not?</td>
</tr>
</tbody>
</table>
1.2.4 Testing your configuration

The best way to test changes to your configuration is with the provided Xephyr script. This will run Qtile with your config.py inside a nested X server and prevent your running instance of Qtile from crashing if something goes wrong. See *Hacking Qtile* for more information on using Xephyr.

1.2.5 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 xsessions 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.

Running from systemd

This case will cover automatic login to Qtile after booting the system without using display manager. It logsins 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.

```bash
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:

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

exec qtile start
```

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`.

```python
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.

```bash
$ cat /usr/share/xsessions/qtile_gnome.desktop
[Desktop Entry]
```

(continues on next page)
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
                   org.gnome.SettingsDaemon.XSettings;
```

Or for older Gnome versions

```
$ cat /usr/share/gnome-session/sessions/qtile.session
[GNOME Session]
Name=Qtile session
RequiredComponents=qtile;org.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).
1.3 Troubleshooting

1.3.1 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`:

```python
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
raise Exception as e:
    logger.exception(e)
```

`logger.warning` is convenient because its messages will always be visible 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.

1.3.2 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:

```bash
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.

1.4 Running Qtile as a Wayland Compositor

Some functionality may not yet be implemented in the Wayland compositor. Please see the discussion here to see the current state of development.
1.4.1 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:

```python
from libqtile import qtile

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

1.4.2 Keyboard Configuration

Keyboard management is done using xkbcommon via the Python bindings. xkbcommon’s initial configuration can be set using environmental variables; see their docs for more information. The `XKB_DEFAULT_X` environmental variables have corresponding settings in X11’s keyboard configuration, so if you have these defined already simply copy their values into these variables, otherwise see X11’s helpful XKB guide to see the syntax for these settings. Simply set these variables before starting Qtile and the initial keyboard state will match these settings.

If you want to change keyboard configuration during runtime, you can use the core’s `set_keymap` command (see Core Commands below).

1.4.3 Running X11-Only Programs

Qtile _does_ support XWayland. This 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.

1.4.4 Core Commands

Core

```python
class libqtile.backend.wayland.core.Core
    cmd_change_vt(vt: int) → bool
    Change virtual terminal to that specified

    cmd_commands() → List[str]
    Returns a list of possible commands for this object
    Used by __qsh__ for command completion and online help

    cmd_doc(name) → str
    Returns the documentation for a specified command name
    Used by __qsh__ to provide online help.

    cmd_eval(code: str) → Tuple[bool, Optional[str]]
    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.

    cmd_function(function, *args, **kwargs) → None
    Call a function with current object as argument
```
**cmd_info()** → Dict
Get basic information about the running backend.

**cmd_items(name)** → Tuple[bool, Optional[List[Union[str, int]]]]
Returns a list of contained items for the specified name
Used by __qsh__ to allow navigation of the object graph.

**cmd_set_keymap(layout: Optional[str] = None, options: Optional[str] = None, variant: Optional[str] = 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.

### 1.5 Shell commands

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.

#### 1.5.1 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.

#### 1.5.2 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.

**Navigating the Object Graph**

The shell presents a filesystem-like interface to the object graph - the built-in "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/

> cd bar
Item required for bar

> ls bar
bar[bottom]/

> cd bar/bottom
bar['bottom']>

Please refer to Keys 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'])
```

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]>
```

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
```

(continues on next page)
1.5.3 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.

How it works

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

As per Commands API, Qtile's object 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 (NB the root node is called cmd when using the cmd-obj script to give it an addressable name).

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 Keys for more information.

Note: You will see from the graph on Commands API 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.

**Examples:**

**Output of qtile cmd-obj -h**

```
usage: qtile cmd-obj [-h] [--object OBJ_SPEC [OBJ_SPEC ...]]
                [-f 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 cmd
qtile cmd-obj -o cmd -f prev_layout -i
qtile cmd-obj -o cmd -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 -o cmd -f restart # restart qtile
```

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

```
-o group 3 -f commands
-o group 3 -f doc
-o group 3 -f eval
-o group 3 -f focus_back
-o group 3 -f focus_by_name
-o group 3 -f function

Returns a list of possible commands for this object
* Returns the documentation for a specified command name
* Evaluates code in the same context as this function
Focus the window that had focus before the current one..
* Focus the first window with the given name. Do nothing..
* Call a function with current object as argument
```

(continues on next page)
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-o group 3 -f info</code></td>
<td>Returns a dictionary of info for this group</td>
</tr>
<tr>
<td><code>-o group 3 -f info_by_name</code></td>
<td>Get the info for the first window with the given name, without giving it</td>
</tr>
<tr>
<td><code>-o group 3 -f items</code></td>
<td>Returns a list of contained items for the specified name</td>
</tr>
<tr>
<td><code>-o group 3 -f next_window</code></td>
<td>Focus the next window in group.</td>
</tr>
<tr>
<td><code>-o group 3 -f prev_window</code></td>
<td>Focus the previous window in group.</td>
</tr>
<tr>
<td><code>-o group 3 -f set_label</code></td>
<td>Set the display name of current group to be used inGroupBox widget.</td>
</tr>
<tr>
<td><code>-o group 3 -f setlayout</code></td>
<td></td>
</tr>
<tr>
<td><code>-o group 3 -f switch_groups</code></td>
<td>Switch position of current group with name</td>
</tr>
<tr>
<td><code>-o group 3 -f toscreen</code></td>
<td>Pull a group to a specified screen.</td>
</tr>
<tr>
<td><code>-o group 3 -f unminimize_all</code></td>
<td>Unminimise all windows in this group.</td>
</tr>
</tbody>
</table>

Output of `qtile cmd-obj -o cmd`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-o cmd -f add_rule</code></td>
<td>Add a dgroup rule, returns rule_id needed to remove it</td>
</tr>
<tr>
<td><code>-o cmd -f addgroup</code></td>
<td>Add a group with the given name</td>
</tr>
<tr>
<td><code>-o cmd -f commands</code></td>
<td>Returns a list of possible commands for this object</td>
</tr>
<tr>
<td><code>-o cmd -f critical</code></td>
<td>Set log level to CRITICAL</td>
</tr>
<tr>
<td><code>-o cmd -f debug</code></td>
<td>Set log level to DEBUG</td>
</tr>
<tr>
<td><code>-o cmd -f delgroup</code></td>
<td>Delete a group with the given name</td>
</tr>
<tr>
<td><code>-o cmd -f display_kb</code></td>
<td>Display table of key bindings</td>
</tr>
<tr>
<td><code>-o cmd -f doc</code></td>
<td>Returns the documentation for a specified command name</td>
</tr>
<tr>
<td><code>-o cmd -f error</code></td>
<td>Set log level to ERROR</td>
</tr>
<tr>
<td><code>-o cmd -f eval</code></td>
<td>Evaluates code in the same context as this function</td>
</tr>
<tr>
<td><code>-o cmd -f findwindow</code></td>
<td>Launch prompt widget to find a window of the given name</td>
</tr>
<tr>
<td><code>-o cmd -f focus_by_click</code></td>
<td>Bring a window to the front</td>
</tr>
<tr>
<td><code>-o cmd -f function</code></td>
<td>Call a function with current object as argument</td>
</tr>
<tr>
<td><code>-o cmd -f get_info</code></td>
<td>Prints info for all groups</td>
</tr>
<tr>
<td><code>-o cmd -f get_state</code></td>
<td>Get pickled state for restarting qtile</td>
</tr>
<tr>
<td><code>-o cmd -f get_test_data</code></td>
<td>Returns any content arbitrarily set in the self.test_data attribute.</td>
</tr>
<tr>
<td><code>-o cmd -f groups</code></td>
<td>Return a dictionary containing information for all groups</td>
</tr>
<tr>
<td><code>-o cmd -f hide_show_bar</code></td>
<td>Toggle visibility of a given bar</td>
</tr>
<tr>
<td><code>-o cmd -f info</code></td>
<td>Set log level to INFO</td>
</tr>
<tr>
<td><code>-o cmd -f internal_windows</code></td>
<td>Return info for each internal window (bars, for example)</td>
</tr>
<tr>
<td><code>-o cmd -f items</code></td>
<td>Returns a list of contained items for the specified name</td>
</tr>
<tr>
<td><code>-o cmd -f list_widgets</code></td>
<td>List of all addressible widget names</td>
</tr>
<tr>
<td><code>-o cmd -f next_layout</code></td>
<td>Switch to the next layout.</td>
</tr>
<tr>
<td><code>-o cmd -f next_screen</code></td>
<td>Move to next screen</td>
</tr>
<tr>
<td><code>-o cmd -f next_urgent</code></td>
<td>Focus next window with urgent hint</td>
</tr>
<tr>
<td><code>-o cmd -f pause</code></td>
<td>Drops into pdb</td>
</tr>
<tr>
<td><code>-o cmd -f prev_layout</code></td>
<td>Switch to the previous layout.</td>
</tr>
<tr>
<td><code>-o cmd -f prev_screen</code></td>
<td>Move to the previous screen</td>
</tr>
<tr>
<td><code>-o cmd -f qtile_info</code></td>
<td>Returns a dictionary of info on the Qtile instance</td>
</tr>
<tr>
<td><code>-o cmd -f qtilecmd</code></td>
<td>Execute a Qtile command using the client syntax</td>
</tr>
</tbody>
</table>

1.5. Shell commands
-o cmd -f remove_rule  * Remove a dgroup rule by rule_id
-0 cmd -f restart     * Restart qtile
-0 cmd -f run_extension  * Run extensions
-0 cmd -f run_external  * Run external Python script
-0 cmd -f screens     * Return a list of dictionaries providing information on
  → all screens
-0 cmd -f shutdown    * Quit Qtile
-0 cmd -f simulate_keypress * Simulates a keypress on the focused window.
-0 cmd -f spawn       * Run cmd in a shell.
-0 cmd -f spawncmd    * Spawn a command using a prompt widget, with tab-
  → completion.
-0 cmd -f status      * Return "OK" if Qtile is running
-0 cmd -f switch_groups * Switch position of groupa to groupb
-0 cmd -f switchgroup  * Launch prompt widget to switch to a given group to the
  → current screen
-0 cmd -f sync        * Sync the X display. Should only be used for development
-0 cmd -f to_layout_index * Switch to the layout with the given index in self.
  → layouts.
-0 cmd -f to_screen   * Warp focus to screen n, where n is a 0-based screen
  → number
-0 cmd -f togroup     * Launch prompt widget to move current window to a given
  → group
-0 cmd -f tracemalloc_dump Dump tracemalloc snapshot
-0 cmd -f tracemalloc_toggle Toggle tracemalloc status
-0 cmd -f warning     Set log level to WARNING
-0 cmd -f windows     * Return info for each client window

1.5.4 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

1.5.5 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()")
1.5.6 dqtile-cmd

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

Examples:

Output of dqtile-cmd -o cmd

```
-dmenu:
  Alt-l  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. Accepts 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.
```

(continues on next page)
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 use --force-dmenu as the first argument.

1.5.7 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.

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:

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

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

Installing and Running the Kernel

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

```bash
$ 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:

```bash
$ 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:

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

The iqshell script will launch a Jupyter terminal console with the qshell kernel.
**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).
2.1 Built-in Extensions

2.1.1 CommandSet

```python
class libqtile.extension.CommandSet(**config)

Give list of commands to be executed in dmenu style.

ex. manage mocp daemon:

```python
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)),
```

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>defines the normal background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>command</td>
<td>None</td>
<td>the command to be launched (string or list with arguments)</td>
</tr>
<tr>
<td>commands</td>
<td>None</td>
<td>dictionary of commands where key is runnable command</td>
</tr>
<tr>
<td>dmenu_bottom</td>
<td>False</td>
<td>dmenu appears at the bottom of the screen</td>
</tr>
<tr>
<td>dmenu_command</td>
<td>'dmenu'</td>
<td>the dmenu command to be launched</td>
</tr>
<tr>
<td>dmenu_font</td>
<td>None</td>
<td>override the default 'font' and 'fontsize' options for dmenu</td>
</tr>
<tr>
<td>dmenu_height</td>
<td>None</td>
<td>defines the height (only supported by some dmenu forks)</td>
</tr>
<tr>
<td>dmenu_ignorecase</td>
<td>False</td>
<td>dmenu matches menu items case insensitively</td>
</tr>
<tr>
<td>dmenu_lines</td>
<td>None</td>
<td>dmenu lists items vertically, with the given number of lines</td>
</tr>
<tr>
<td>dmenu_prompt</td>
<td>None</td>
<td>defines the prompt to be displayed to the left of the input field</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>defines the font name to be used</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>defines the font size to be used</td>
</tr>
<tr>
<td>foreground</td>
<td>None</td>
<td>defines the normal foreground color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>pre_commands</td>
<td>None</td>
<td>list of commands to be executed before getting dmenu answer</td>
</tr>
<tr>
<td>selected_background</td>
<td>None</td>
<td>defines the selected background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_foreground</td>
<td>None</td>
<td>defines the selected foreground color (#RGB or #RRGGBB)</td>
</tr>
</tbody>
</table>
2.1.2 Dmenu

```
class libqtile.extension.Dmenu(**config)
Python wrapper for dmenu http://tools.suckless.org/dmenu/
```

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>defines the normal background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>command</td>
<td>None</td>
<td>the command to be launched (string or list with arguments)</td>
</tr>
<tr>
<td>dmenu_bottom</td>
<td>False</td>
<td>dmenu appears at the bottom of the screen</td>
</tr>
<tr>
<td>dmenu_command</td>
<td>'dmenu'</td>
<td>the dmenu command to be launched</td>
</tr>
<tr>
<td>dmenu_font</td>
<td>None</td>
<td>override the default 'font' and 'fontsize' options for dmenu</td>
</tr>
<tr>
<td>dmenu_height</td>
<td>None</td>
<td>defines the height (only supported by some dmenu forks)</td>
</tr>
<tr>
<td>dmenu_ignorecase</td>
<td>False</td>
<td>dmenu matches menu items case insensitively</td>
</tr>
<tr>
<td>dmenu_lines</td>
<td>None</td>
<td>dmenu lists items vertically, with the given number of lines</td>
</tr>
<tr>
<td>dmenu_prompt</td>
<td>None</td>
<td>defines the prompt to be displayed to the left of the input field</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>defines the font name to be used</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>defines the font size to be used</td>
</tr>
<tr>
<td>foreground</td>
<td>None</td>
<td>defines the normal foreground color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_background</td>
<td>None</td>
<td>defines the selected background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_foreground</td>
<td>None</td>
<td>defines the selected foreground color (#RGB or #RRGGBB)</td>
</tr>
</tbody>
</table>

2.1.3 DmenuRun

```
class libqtile.extension.DmenuRun(**config)
Special case to run applications.
```

```
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
    ))),
]```
### 2.1.4 J4DmenuDesktop

class libqtile.extension.J4DmenuDesktop(**config)

Python wrapper for j4-dmenu-desktop [https://github.com/enkore/j4-dmenu-desktop](https://github.com/enkore/j4-dmenu-desktop)

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>defines the normal background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>command</td>
<td>None</td>
<td>the command to be launched (string or list with arguments)</td>
</tr>
<tr>
<td>dmenu_bottom</td>
<td>False</td>
<td>dmenu appears at the bottom of the screen</td>
</tr>
<tr>
<td>dmenu_command</td>
<td>'dmenu'</td>
<td>the dmenu command to be launched</td>
</tr>
<tr>
<td>dmenu_font</td>
<td>None</td>
<td>override the default 'font' and 'fontsize' options for dmenu</td>
</tr>
<tr>
<td>dmenu_height</td>
<td>None</td>
<td>defines the height (only supported by some dmenu forks)</td>
</tr>
<tr>
<td>dmenu_ignorecase</td>
<td>False</td>
<td>dmenu matches menu items case insensitively</td>
</tr>
<tr>
<td>dmenu_lines</td>
<td>None</td>
<td>dmenu lists items vertically, with the given number of lines</td>
</tr>
<tr>
<td>dmenu_prompt</td>
<td>None</td>
<td>defines the prompt to be displayed to the left of the input field</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>defines the font name to be used</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>defines the font size to be used</td>
</tr>
<tr>
<td>foreground</td>
<td>None</td>
<td>defines the normal foreground color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_background</td>
<td>None</td>
<td>defines the selected background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_foreground</td>
<td>None</td>
<td>defines the selected foreground color (#RGB or #RRGGBB)</td>
</tr>
</tbody>
</table>

```python
import libqtile.extension

class J4DmenuDesktop:
    def __init__(self, **config):
        self.j4dmenu_command = 'j4-dmenu-desktop'
        self.j4dmenu_display_binary = True
        self.j4dmenu_terminal = None
        self.j4dmenu_use_xdg_desktop = True
        self.selected_background = None
        self.selected_foreground = None
```

---

2.1. Built-in Extensions 43
### 2.1.5 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.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>defines the normal background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>command</td>
<td>None</td>
<td>the command to be launched (string or list with arguments)</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>defines the font name to be used</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>defines the font size to be used</td>
</tr>
<tr>
<td>foreground</td>
<td>None</td>
<td>defines the normal foreground color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_background</td>
<td>None</td>
<td>defines the selected background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_foreground</td>
<td>None</td>
<td>defines the selected foreground color (#RGB or #RRGGBB)</td>
</tr>
</tbody>
</table>

### 2.1.6 WindowList

**class** `libqtile.extension.WindowList(**config)`

Give vertical list of all open windows in dmenu. Switch to selected.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_groups</td>
<td>True</td>
<td>If True, list windows from all groups; otherwise only from the current group</td>
</tr>
<tr>
<td>background</td>
<td>None</td>
<td>defines the normal background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>command</td>
<td>None</td>
<td>the command to be launched (string or list with arguments)</td>
</tr>
<tr>
<td>dmenu_bottom</td>
<td>False</td>
<td>dmenu appears at the bottom of the screen</td>
</tr>
<tr>
<td>dmenu_command</td>
<td>'dmenu'</td>
<td>the dmenu command to be launched</td>
</tr>
<tr>
<td>dmenu_font</td>
<td>None</td>
<td>override the default 'font' and 'fontsize' options for dmenu</td>
</tr>
<tr>
<td>dmenu_height</td>
<td>None</td>
<td>defines the height (only supported by some dmenu forks)</td>
</tr>
<tr>
<td>dmenu_ignorecase</td>
<td>False</td>
<td>dmenu matches menu items case insensitively</td>
</tr>
<tr>
<td>dmenu_lines</td>
<td>'80'</td>
<td>Give lines vertically. Set to None get inline</td>
</tr>
<tr>
<td>dmenu_prompt</td>
<td>None</td>
<td>defines the prompt to be displayed to the left of the input field</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>defines the font name to be used</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>defines the font size to be used</td>
</tr>
<tr>
<td>foreground</td>
<td>None</td>
<td>defines the normal foreground color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>item_format</td>
<td>'{group}.{id}:'</td>
<td>the format for the menu items</td>
</tr>
<tr>
<td>selected_background</td>
<td>None</td>
<td>defines the selected background color (#RGB or #RRGGBB)</td>
</tr>
<tr>
<td>selected_foreground</td>
<td>None</td>
<td>defines the selected foreground color (#RGB or #RRGGBB)</td>
</tr>
</tbody>
</table>
## 2.2 Built-in Hooks

**subscribe.addgroup(func)**
Called when group is added

**Arguments**
- name of new group

**subscribe.changegroup(func)**
Called whenever a group change occurs

**Arguments**
None

**subscribe.client_focus(func)**
Called whenever focus moves to a client window

**Arguments**
- Window object of the new focus.

**subscribe.client_killed(func)**
Called after a client has been unmanaged

**Arguments**
- Window object of the killed window.

**subscribe.client_managed(func)**
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

**subscribe.client_mouse_enter(func)**
Called when the mouse enters a client

**Arguments**
- Window of window entered

**subscribe.client_name_updated(func)**
Called when the client name changes

**Arguments**
- Window of client with updated name

**subscribe.client_new(func)**
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
Examples

```python
@libqtile.hook.subscribe.client_new
def func(c):
    if c.name == "xterm":
        c.togroup("a")
    elif c.name == "dzen":
        c.cmd_static(0)
```

subscribe.client_urgent_hint_changed(func)
Called when the client urgent hint changes

Arguments

• Window of client with hint change

subscribe.current_screen_change(func)
Called when the current screen (i.e. the screen with focus) changes

Arguments

None

subscribe.delgroup(func)
Called when group is deleted

Arguments

• name of deleted group

subscribe.enter_chord(func)
Called when key chord begins

Arguments

• name of chord(mode)

subscribe.float_change(func)
Called when a change in float state is made

Arguments

None

subscribe.focus_change(func)
Called when focus is changed, including moving focus between groups or when focus is lost completely

Arguments

None

subscribe.group_window_add(func)
Called when a new window is added to a group

Arguments

• Group receiving the new window

• Window added to the group

subscribe.layout_change(func)
Called on layout change

Arguments

• layout object for new layout
• group object on which layout is changed

```
subscribe.leave_chord(func)
```
Called when key chord ends

**Arguments**

None

```
subscribe.net_wm_icon_change(func)
```
Called on `_NET_WM_ICON` change

**Arguments**

• Window of client with changed icon

```
subscribe.restart(func)
```
Called before qtile is restarted

**Arguments**

None

```
subscribe.screen_change(func)
```
Called when the output configuration is changed (e.g. via randr in X11).

**Arguments**

• `xproto.randr.ScreenChangeNotify` event (X11) or None (Wayland).

```
subscribe.screens_reconfigured(func)
```
Called once `qtile.cmd_reconfigure_screens` has completed (e.g. if `reconfigure_screens` is set to True in your config).

**Arguments**

None

```
subscribe.selection_change(func)
```
Called on selection change

**Arguments**

• name of the selection

• dictionary describing selection, containing `owner` and `selection` as keys

```
subscribe.selection_notify(func)
```
Called on selection notify

**Arguments**

• name of the selection

• dictionary describing selection, containing `owner` and `selection` as keys

```
subscribe.setgroup(func)
```
Called when group is changed

**Arguments**

None

```
subscribe.shutdown(func)
```
Called before qtile is shutdown

**Arguments**

None

### 2.2. Built-in Hooks
None

**subscribe.startup(func)**
Called when qtile is started

**Arguments**
None

**subscribe.startup_complete(func)**
Called when qtile is started after all resources initialized

**Arguments**
None

**subscribe.startup_once(func)**
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

## 2.3 Built-in Layouts

### 2.3.1 Floating

**class** `libqtile.layout.floating.Floating(float_rules: Optional[List[libqtile.config.Match]] = None, no_reposition_rules=None, **config)`

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border_focus</td>
<td>'#0000ff'</td>
<td>Border colour(s) for the focused window.</td>
</tr>
<tr>
<td>border_normal</td>
<td>'#000000'</td>
<td>Border colour(s) for un-focused windows.</td>
</tr>
<tr>
<td>border_width</td>
<td>1</td>
<td>Border width.</td>
</tr>
<tr>
<td>fullscreen_border_width</td>
<td>0</td>
<td>Border width for fullscreen.</td>
</tr>
<tr>
<td>max_border_width</td>
<td>0</td>
<td>Border width for maximize.</td>
</tr>
</tbody>
</table>

### 2.3.2 Bsp

**class** `libqtile.layout.bsp.Bsp(**config)`

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()),

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border_focus</td>
<td>'#881111'</td>
<td>BorderColour(s) for the focused window.</td>
</tr>
<tr>
<td>border_normal</td>
<td>'#220000'</td>
<td>BorderColour(s) for un-focused windows.</td>
</tr>
<tr>
<td>border_width</td>
<td>2</td>
<td>Border width.</td>
</tr>
<tr>
<td>fair</td>
<td>True</td>
<td>New clients are inserted in the shortest branch.</td>
</tr>
<tr>
<td>grow_amount</td>
<td>10</td>
<td>Amount by which to grow a window/column.</td>
</tr>
<tr>
<td>lower_right</td>
<td>True</td>
<td>New client occupies lower or right subspace.</td>
</tr>
<tr>
<td>margin</td>
<td>0</td>
<td>Margin of the layout (int or list of ints [N E S W]).</td>
</tr>
<tr>
<td>ratio</td>
<td>1.6</td>
<td>Width/height ratio that defines the partition direction.</td>
</tr>
</tbody>
</table>

### 2.3.3 Columns

class libqtile.layout.columns.Columns(**config)**

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, splitting 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:

```python
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()),

(continues on next page)
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()),

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border_focus</td>
<td>'#881111'</td>
<td>Border colour(s) for the focused window.</td>
</tr>
<tr>
<td>border_focus_stack</td>
<td>'#881111'</td>
<td>Border colour(s) for the focused window in stacked columns.</td>
</tr>
<tr>
<td>border_normal</td>
<td>'#220000'</td>
<td>Border colour(s) for un-focused windows.</td>
</tr>
<tr>
<td>border_normal_stack</td>
<td>'#220000'</td>
<td>Border colour(s) for un-focused windows in stacked columns.</td>
</tr>
<tr>
<td>border_on_single</td>
<td>False</td>
<td>Draw a border when there is one only window.</td>
</tr>
<tr>
<td>border_width</td>
<td>2</td>
<td>Border width.</td>
</tr>
<tr>
<td>fair</td>
<td>False</td>
<td>Add new windows to the column with least windows.</td>
</tr>
<tr>
<td>grow_amount</td>
<td>10</td>
<td>Amount by which to grow a window/column.</td>
</tr>
<tr>
<td>insert_position</td>
<td>0</td>
<td>Position relative to the current window where new ones are inserted (0 means right above the current window, 1 means right after).</td>
</tr>
<tr>
<td>margin</td>
<td>0</td>
<td>Margin of the layout (int or list of ints [N E S W]).</td>
</tr>
<tr>
<td>margin_on_single</td>
<td>None</td>
<td>Margin when only one window. (int or list of ints [N E S W])</td>
</tr>
<tr>
<td>num_columns</td>
<td>2</td>
<td>Preferred number of columns.</td>
</tr>
<tr>
<td>split</td>
<td>True</td>
<td>New columns presentation mode.</td>
</tr>
<tr>
<td>wrap_focus_columns</td>
<td>True</td>
<td>Wrap the screen when moving focus across columns.</td>
</tr>
<tr>
<td>wrap_focus_rows</td>
<td>True</td>
<td>Wrap the screen when moving focus across rows.</td>
</tr>
<tr>
<td>wrap_focus_stacks</td>
<td>True</td>
<td>Wrap the screen when moving focus across stacked.</td>
</tr>
</tbody>
</table>

### 2.3.4 Matrix

**class** libqtile.layout.matrix.Matrix(columns=2, **config)**

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.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>border_focus</td>
<td>'#0000ff'</td>
<td>Border colour(s) for the focused window.</td>
</tr>
<tr>
<td>border_normal</td>
<td>'#000000'</td>
<td>Border colour(s) for un-focused windows.</td>
</tr>
<tr>
<td>border_width</td>
<td>1</td>
<td>Border width.</td>
</tr>
<tr>
<td>margin</td>
<td>0</td>
<td>Margin of the layout (int or list of ints [N E S W])</td>
</tr>
</tbody>
</table>
2.3.5 Max

```python
class libqtile.layout.max.Max(**config)
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.

2.3.6 MonadTall

```python
class libqtile.layout.xmonad.MonadTall(**config)
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 cmd_grow_main and cmd_shrink_main or, while the main pane is in focus, cmd_grow and cmd_shrink.

Using the cmd_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 cmd_grow and cmd_shrink methods. The other secondary panes will adjust their sizes to smoothly fill all of the space.
Panes can be moved with the `cmd_shuffle_up` and `cmd_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".

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

Normalizing/Resetting:

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

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

Maximizing:

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

Suggested Bindings:

```python
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.normalize()),
Key([modkey], "o", lazy.layout.maximize()),
Key([modkey, "shift"], "space", lazy.layout.flip()),
```
### MonadWide

**class** `libqtile.layout.xmonad.MonadWide(**config)**

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 `cmd_grow_main` and `cmd_shrink_main` or, while the main pane is in focus, `cmd_grow` and `cmd_shrink`.

```
---------------------
|                    |
|                    |
|___________________|
| |                 |
| |                 |
| |_________________|

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

```

(continues on next page)
Secondary-panes:

Occupyng 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 cmd_grow and cmd_shrink methods. The other secondary panes will adjust their sizes to smoothly fill all of the space.

Panes can be moved with the cmd_shuffle_up and cmd_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.

Normalizing/Resetting:

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

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

Maximizing:

To toggle a client window between its minimum and maximum sizes simply use the cmd_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.normalize()),
```
Key([modkey], "o", lazy.layout.maximize()),
Key([modkey, "shift"], "space", lazy.layout.flip()),

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>align</td>
<td>0</td>
<td>Which side master plane will be placed (one of MonadTall._left or MonadTall._right)</td>
</tr>
<tr>
<td>border_focus</td>
<td>'#ff0000'</td>
<td>Border colour(s) for the focused window.</td>
</tr>
<tr>
<td>border_normal</td>
<td>'#000000'</td>
<td>Border colour(s) for un-focused windows.</td>
</tr>
<tr>
<td>border_width</td>
<td>2</td>
<td>Border width.</td>
</tr>
<tr>
<td>change_ratio</td>
<td>0.05</td>
<td>Resize ratio</td>
</tr>
<tr>
<td>change_size</td>
<td>20</td>
<td>Resize change in pixels</td>
</tr>
<tr>
<td>margin</td>
<td>0</td>
<td>Margin of the layout</td>
</tr>
<tr>
<td>max_ratio</td>
<td>0.75</td>
<td>The percent of the screen-space the master pane should occupy at maximum.</td>
</tr>
<tr>
<td>min_ratio</td>
<td>0.25</td>
<td>The percent of the screen-space the master pane should occupy at minimum.</td>
</tr>
<tr>
<td>min_secondary_size</td>
<td>85</td>
<td>minimum size in pixel for a secondary pane window</td>
</tr>
<tr>
<td>new_client_position</td>
<td>'after_current'</td>
<td>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,</td>
</tr>
<tr>
<td>ratio</td>
<td>0.5</td>
<td>The percent of the screen-space the master pane should occupy by default.</td>
</tr>
<tr>
<td>single_border_width</td>
<td>None</td>
<td>Border width for single window</td>
</tr>
<tr>
<td>single_margin</td>
<td>None</td>
<td>Margin size for single window</td>
</tr>
</tbody>
</table>

### 2.3.8 RatioTile

**class libqtile.layout.ratiotile.RatioTile(***config**)

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

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

class libqtile.layout.slice.Slice(**config)

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fallback</td>
<td><code>&lt;libqtile.layout.max. Max object at 0x7fcf58b478d0&gt;</code></td>
<td>Layout to be used for the non-slice area.</td>
</tr>
<tr>
<td>match</td>
<td>None</td>
<td>Match-object describing which window(s) to move to the slice.</td>
</tr>
<tr>
<td>side</td>
<td>'left'</td>
<td>Position of the slice (left, right, top, bottom).</td>
</tr>
<tr>
<td>width</td>
<td>256</td>
<td>Slice width.</td>
</tr>
</tbody>
</table>

2.3.10 Stack

class libqtile.layout.stack.Stack(**config)

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.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autosplit</td>
<td>False</td>
<td>Auto split all new stacks.</td>
</tr>
<tr>
<td>border_focus</td>
<td>'#0000ff'</td>
<td>Border colour(s) for the focused window.</td>
</tr>
<tr>
<td>border_normal</td>
<td>'#000000'</td>
<td>Border colour(s) for un-focused windows.</td>
</tr>
<tr>
<td>border_width</td>
<td>1</td>
<td>Border width.</td>
</tr>
<tr>
<td>fair</td>
<td>False</td>
<td>Add new windows to the stacks in a round robin way.</td>
</tr>
<tr>
<td>margin</td>
<td>0</td>
<td>Margin of the layout (int or list of ints [N E S W])</td>
</tr>
<tr>
<td>num_stacks</td>
<td>2</td>
<td>Number of stacks.</td>
</tr>
</tbody>
</table>

2.3.11 Tile

class libqtile.layout.tile.Tile(**config)

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

### 2.3.12 TreeTab

**class** `libqtile.layout.tree.TreeTab(**config)`

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   |
+------------+
| 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   |
```

---

2.3. Built-in Layouts

57
<table>
<thead>
<tr>
<th>Section Foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window A</td>
</tr>
<tr>
<td>Window B</td>
</tr>
<tr>
<td>Window C</td>
</tr>
<tr>
<td>-------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Section Bar</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active_bg</td>
<td>'000080'</td>
<td>Background color of active tab</td>
</tr>
<tr>
<td>active_fg</td>
<td>'ffffff'</td>
<td>Foreground color of active tab</td>
</tr>
<tr>
<td>bg_color</td>
<td>'000000'</td>
<td>Background color of tabs</td>
</tr>
<tr>
<td>border_width</td>
<td>2</td>
<td>Width of the border</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Font</td>
</tr>
<tr>
<td>fontsize</td>
<td>14</td>
<td>Font pixel size</td>
</tr>
<tr>
<td>inactive_bg</td>
<td>'606060'</td>
<td>Background color of inactive tab</td>
</tr>
<tr>
<td>inactive_fg</td>
<td>'ffffff'</td>
<td>Foreground color of inactive tab</td>
</tr>
<tr>
<td>level_shift</td>
<td>8</td>
<td>Shift for children tabs</td>
</tr>
<tr>
<td>margin_left</td>
<td>6</td>
<td>Left margin of tab panel</td>
</tr>
<tr>
<td>margin_y</td>
<td>6</td>
<td>Vertical margin of tab panel</td>
</tr>
<tr>
<td>padding_left</td>
<td>6</td>
<td>Left padding for tabs</td>
</tr>
<tr>
<td>padding_x</td>
<td>6</td>
<td>Left padding for tab label</td>
</tr>
<tr>
<td>padding_y</td>
<td>2</td>
<td>Top padding for tab label</td>
</tr>
<tr>
<td>panel_width</td>
<td>150</td>
<td>Width of the left panel</td>
</tr>
<tr>
<td>place_right</td>
<td>False</td>
<td>Place the tab panel on the right side</td>
</tr>
<tr>
<td>previous_on_rm</td>
<td>False</td>
<td>Focus previous window on close instead of first.</td>
</tr>
<tr>
<td>section_bottom</td>
<td>6</td>
<td>Bottom margin of section</td>
</tr>
<tr>
<td>section_fg</td>
<td>'ffffff'</td>
<td>Color of section label</td>
</tr>
<tr>
<td>section_fontsize</td>
<td>11</td>
<td>Font pixel size of section label</td>
</tr>
<tr>
<td>section_left</td>
<td>4</td>
<td>Left margin of section label</td>
</tr>
<tr>
<td>section_padding</td>
<td>4</td>
<td>Bottom of margin section label</td>
</tr>
<tr>
<td>section_top</td>
<td>4</td>
<td>Top margin of section label</td>
</tr>
<tr>
<td>sections</td>
<td>['Default']</td>
<td>Foreground color of inactive tab</td>
</tr>
<tr>
<td>urgent_bg</td>
<td>'ff0000'</td>
<td>Background color of urgent tab</td>
</tr>
<tr>
<td>urgent_fg</td>
<td>'ffffff'</td>
<td>Foreground color of urgent tab</td>
</tr>
<tr>
<td>vspace</td>
<td>2</td>
<td>Space between tabs</td>
</tr>
</tbody>
</table>
2.3.13 VerticalTile

**class** libqtile.layout.verticaltile.VerticalTile(**config**)

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.

```
----------------- ----------------- ---
| | | | | |
| 1 | <-- Panes | | |
| | | | | |
|---------------| | | | |
| | | | | |
| 2 | <-----+ | 1 | | Master Area
| | | | | |
|---------------| | | | |
| | | | | |
| 3 | <-----| |
| | | | | |
|---------------| | | | |
| | | | | |
| 4 | <-----+ | 2 | | Secondary Area
| | | | | |
|---------------| | | | |
| | | | 2 | |
| 5 | <-----+ |---------------| | |
| | | 3 | |
|---------------| | | | |
```

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

```
In some cases VerticalTile can be useful on horizontal mounted monitors two. For example if you want to have a webbrowser and a shell below it.
```

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()),
```
### 2.3.14 Zoomy

**class** `libqtile.layout.zoomy.Zoomy(**config)**

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

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

### 2.4 Built-in Widgets

#### 2.4.1 AGroupBox

**class** `libqtile.widget.AGroupBox(**config)**

A widget that graphically displays the current group.

Supported bar orientations: horizontal only.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>border</td>
<td>'000000'</td>
<td>Group box border color</td>
</tr>
<tr>
<td>borderwidth</td>
<td>3</td>
<td>Current group border width</td>
</tr>
<tr>
<td>center_aligned</td>
<td>True</td>
<td>Center-aligned group box</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>margin</td>
<td>3</td>
<td>Margin inside the box</td>
</tr>
<tr>
<td>margin_x</td>
<td>None</td>
<td>X Margin. Overrides 'margin' if set</td>
</tr>
<tr>
<td>margin_y</td>
<td>None</td>
<td>Y Margin. Overrides 'margin' if set</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>padding_x</td>
<td>None</td>
<td>X Padding. Overrides 'padding' if set</td>
</tr>
<tr>
<td>padding_y</td>
<td>None</td>
<td>Y Padding. Overrides 'padding' if set</td>
</tr>
</tbody>
</table>
2.4.2 Backlight

class libqtile.widget.Backlight(**config)

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. Depending on the setup, the user may need to be added to the video group to have permission to write to this interface. This depends on having the correct udev rules the brightness file; these are typically installed alongside brightness tools such as brightnessctl (which changes the group to 'video') so installing that is an easy way to get it working.

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

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>backlight_name</td>
<td>'acpi_video0'</td>
<td>ACPI name of a backlight device</td>
</tr>
<tr>
<td>brightness_file</td>
<td>'brightness'</td>
<td>Name of file with the current brightness in /sys/class/backlight/backlight_name</td>
</tr>
<tr>
<td>change_command</td>
<td>'xbacklight -set {0}'</td>
<td>Execute command to change value</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'{percent:2.0%}'</td>
<td>Display format</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_brightness_file</td>
<td>'max_brightness'</td>
<td>Name of file with the maximum brightness in /sys/class/backlight/backlight_name</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>step</td>
<td>10</td>
<td>Percent of backlight every scroll changed</td>
</tr>
<tr>
<td>update_interval</td>
<td>0.2</td>
<td>The delay in seconds between updates</td>
</tr>
</tbody>
</table>
## 2.4.3 Battery

**class** `libqtile.widget.Battery(**config)**

A text-based battery monitoring widget currently supporting FreeBSD

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>battery</td>
<td>0</td>
<td>Which battery should be monitored (battery number or name)</td>
</tr>
<tr>
<td>charge_char</td>
<td>'^'</td>
<td>Character to indicate the battery is charging</td>
</tr>
<tr>
<td>discharge_char</td>
<td>'V'</td>
<td>Character to indicate the battery is discharging</td>
</tr>
<tr>
<td>empty_char</td>
<td>'x'</td>
<td>Character to indicate the battery is empty</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'{char}'</td>
<td>Display format</td>
</tr>
<tr>
<td>full_char</td>
<td>'='</td>
<td>Character to indicate the battery is full</td>
</tr>
<tr>
<td>hide_threshold</td>
<td>None</td>
<td>Hide the text when there is enough energy 0 &lt;= x &lt; 1</td>
</tr>
<tr>
<td>low_background</td>
<td>None</td>
<td>Background color on low battery</td>
</tr>
<tr>
<td>low_foreground</td>
<td>'FF0000'</td>
<td>Font color on low battery</td>
</tr>
<tr>
<td>low_percentage</td>
<td>0.1</td>
<td>Indicates when to use the low_foreground color 0 &lt; x &lt; 1</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>notify_below</td>
<td>None</td>
<td>Send a notification below this battery level.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>show_short_text</td>
<td>True</td>
<td>Show &quot;Full&quot; or &quot;Empty&quot; rather than formatted text</td>
</tr>
<tr>
<td>unknown_char</td>
<td>'?'</td>
<td>Character to indicate the battery status is unknown</td>
</tr>
<tr>
<td>update_interval</td>
<td>60</td>
<td>Seconds between status updates</td>
</tr>
</tbody>
</table>

## 2.4.4 BatteryIcon

**class** `libqtile.widget.BatteryIcon(**config)**

Battery life indicator widget.

Supported bar orientations: horizontal only
<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>battery</td>
<td>0</td>
<td>Which battery should be monitored</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>theme_path</td>
<td>'{/home/docs/checkouts/readthedocs.org/user_builds/qtile/checkouts/v0.20.0/libqtile/resources/battery-icons}'</td>
<td>Path of the icons</td>
</tr>
<tr>
<td>update_interval</td>
<td>60</td>
<td>Seconds between status updates</td>
</tr>
</tbody>
</table>

### 2.4.5 Bluetooth

class libqtile.widget.Bluetooth(**config)

Displays bluetooth status or connected device.

Uses dbus to communicate with the system bus.

Widget requirements: dbus-next.

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>hci</td>
<td>'/dev_XX_XX_XX_XX_XX_XX'</td>
<td>hci0 device path, can be found with d-feet or similar dbus explorer.</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
</tbody>
</table>

2.4. Built-in Widgets
2.4.6 CPU

class libqtile.widget.CPU(**config)
A simple widget to display CPU load and frequency.

Widget requirements: psutil.

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground color</td>
</tr>
</tbody>
</table>
| format            | 'CPU
{freq_current}GHz
{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.                    |
| update_interval   | 1.0         | Update interval for the CPU widget               |

2.4.7 CPUGraph

class libqtile.widget.CPUGraph(**config)
Display CPU usage graph.

Widget requirements: psutil.

Supported bar orientations: horizontal only

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

**class** `libqtile.widget.Canto(**config)`

Display RSS feeds updates using the canto console reader

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>all_format</code></td>
<td>'{number}'</td>
<td>All feeds display format</td>
</tr>
<tr>
<td><code>background</code></td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td><code>feeds</code></td>
<td>[]</td>
<td>List of feeds to display, empty for all</td>
</tr>
<tr>
<td><code>fetch</code></td>
<td>False</td>
<td>Whether to fetch new items on update</td>
</tr>
<tr>
<td><code>fmt</code></td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td><code>font</code></td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td><code>fontshadow</code></td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td><code>fontsize</code></td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td><code>foreground</code></td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td><code>markup</code></td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td><code>max_chars</code></td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td><code>mouse_callbacks</code></td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td><code>one_format</code></td>
<td>'{name}: {number}'</td>
<td>One feed display format</td>
</tr>
<tr>
<td><code>padding</code></td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td><code>update_interval</code></td>
<td>600</td>
<td>Update interval in seconds, if none, the widget updates whenever it's done.</td>
</tr>
</tbody>
</table>

## 2.4.9 CapsNumLockIndicator

**class** `libqtile.widget.CapsNumLockIndicator(**config)`

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

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>background</code></td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td><code>fmt</code></td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td><code>font</code></td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td><code>fontshadow</code></td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td><code>fontsize</code></td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td><code>foreground</code></td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td><code>markup</code></td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td><code>max_chars</code></td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td><code>mouse_callbacks</code></td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td><code>padding</code></td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td><code>update_interval</code></td>
<td>0.5</td>
<td>Update Time in seconds.</td>
</tr>
</tbody>
</table>
2.4.10 CheckUpdates

```python
class libqtile.widget.CheckUpdates(**config)
```

Shows number of pending updates in different unix systems.

**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 command/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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>colour_have_updates</td>
<td>ffffff</td>
<td>Colour when there are updates.</td>
</tr>
<tr>
<td>colour_no_updates</td>
<td>ffffff</td>
<td>Colour when there's no updates.</td>
</tr>
<tr>
<td>custom_command</td>
<td>None</td>
<td>Custom shell command for checking updates (counts the lines of the output)</td>
</tr>
<tr>
<td>custom_command_modification</td>
<td></td>
<td>Lambda function to modify line count from custom_command</td>
</tr>
<tr>
<td>display_format</td>
<td>'Updates: {updates}'</td>
<td>Display format if updates available</td>
</tr>
<tr>
<td>distro</td>
<td>'Arch'</td>
<td>Name of your distribution</td>
</tr>
<tr>
<td>execute</td>
<td>None</td>
<td>Command to execute on click</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color; default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>ffffff</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>no_update_string</td>
<td>''</td>
<td>String to display if no updates available</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>restart_indicator</td>
<td>''</td>
<td>Indicator to represent reboot is required. (Ubuntu only)</td>
</tr>
<tr>
<td>update_interval</td>
<td>60</td>
<td>Update interval in seconds.</td>
</tr>
</tbody>
</table>
2.4.11 Chord

class libqtile.widget.Chord(width=CALCULATED, **config)
Display current key chord

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>chords_colors</td>
<td>{}</td>
<td>colors per chord in form of tuple ('bg', 'fg').</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>name_transform</td>
<td>&lt;function Chord. &lt;lambda&gt; at 0x7fcf58672950&gt;</td>
<td>preprocessor for chord name it is pure function string -&gt; string</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
</tbody>
</table>

2.4.12 Clipboard

class libqtile.widget.Clipboard(width=CALCULATED, **config)
Display current clipboard contents

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>blacklist</td>
<td>['keepassx']</td>
<td>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.</td>
</tr>
<tr>
<td>blacklist_text</td>
<td>'***********'</td>
<td>text to display when the wm_class is blacklisted</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>max_width</td>
<td>10</td>
<td>maximum number of characters to display (None for all, useful when width is bar.STRETCH)</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>selection</td>
<td>'CLIPBOARD'</td>
<td>the selection to display(CLIPBOARD or PRIMARY)</td>
</tr>
<tr>
<td>timeout</td>
<td>10</td>
<td>Default timeout (seconds) for display text, None to keep forever</td>
</tr>
</tbody>
</table>

2.4. Built-in Widgets
2.4.13 Clock

class libqtile.widget.Clock(**config)
A simple but flexible text-based clock

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'%H:%M'</td>
<td>A Python datetime format string</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>timezone</td>
<td>None</td>
<td>The timezone to use for this clock, either as string if pytz or dateutil is installed (e.g. &quot;US/Central&quot; or anything in /usr/share/zoneinfo), or as tzinfo (e.g. datetime.timezone.utc). None means the system local timezone and is the default.</td>
</tr>
<tr>
<td>update_interval</td>
<td>1.0</td>
<td>Update interval for the clock</td>
</tr>
</tbody>
</table>

2.4.14 Cmus

class libqtile.widget.Cmus(**config)
A simple Cmus 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.

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

Supported bar orientations: horizontal and vertical
### 2.4.15 Countdown

**class** `libqtile.widget.Countdown(**config)`

A simple countdown timer text widget

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>date</td>
<td>datetime.</td>
<td>The datetime for the end of the countdown</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground color</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions.</td>
</tr>
<tr>
<td>noplay_color</td>
<td>'cecece'</td>
<td>Text colour when not playing.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>play_color</td>
<td>'00ff00'</td>
<td>Text colour when playing.</td>
</tr>
<tr>
<td>update_interval</td>
<td>0.5</td>
<td>Update Time in seconds.</td>
</tr>
</tbody>
</table>

2.4. Built-in Widgets 69
## 2.4.16 CryptoTicker

**class** `libqtile.widget.CryptoTicker(**config)`

A cryptocurrency ticker widget, data provided by the coinbase.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="€")

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>crypto</td>
<td>'BTC'</td>
<td>The cryptocurrency to display.</td>
</tr>
<tr>
<td>currency</td>
<td>''</td>
<td>The baseline currency that the value of the crypto is displayed in.</td>
</tr>
<tr>
<td>data</td>
<td>None</td>
<td>Post Data</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
</tbody>
</table>
| 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                                   |
| symbol          | ''      | The symbol for the baseline currency.           |
| update_interval | 600     | Update interval in seconds, if none, the widget updates whenever it's done. |
| url             | None    | Url                                              |
| user_agent      | 'Qtile' | Set the user agent                               |
| xml             | False   | Is XML?                                         |
2.4.17 CurrentLayout

**class libqtile.widget.CurrentLayout(width=CALCULATED, **config)**

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'fffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
</tbody>
</table>

2.4.18 CurrentLayoutIcon

**class libqtile.widget.CurrentLayoutIcon(**config)**

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 order of icon search is:

- dirs in custom_icon_paths config argument
- ~/.icons
- built-in qtile icons

Supported bar orientations: horizontal only
### 2.4.19 CurrentScreen

**class** `libqtile.widget.CurrentScreen(width=CALCULATED, **config)`

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

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>Key</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active_color</td>
<td>'00ff00'</td>
<td>Color when screen is active</td>
</tr>
<tr>
<td>active_text</td>
<td>'A'</td>
<td>Text displayed when the screen is active</td>
</tr>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>scale</td>
<td>1</td>
<td>Scale factor relative to the bar height. Defaults to 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Key</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inactive_color</td>
<td>'ff0000'</td>
<td>Color when screen is inactive</td>
</tr>
<tr>
<td>inactive_text</td>
<td>'I'</td>
<td>Text displayed when the screen is inactive</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
</tbody>
</table>
2.4.20 DF

**class** libqtile.widget.DF(**config**)  
Disk Free Widget

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

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'{p} ({uf}{m}{r:.0f}%)'</td>
<td>String format (p: partition, s: size, f: free space, uf: user free space, m: measure, r: ratio (uf/s))</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>measure</td>
<td>'G'</td>
<td>Measurement (G, M, B)</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>partition</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>update_interval</td>
<td>60</td>
<td>The update interval.</td>
</tr>
<tr>
<td>visible_on_warn</td>
<td>True</td>
<td>Only display if warning</td>
</tr>
<tr>
<td>warn_color</td>
<td>'ff0000'</td>
<td>Warning color</td>
</tr>
<tr>
<td>warn_space</td>
<td>2</td>
<td>Warning space in scale defined by the measure option.</td>
</tr>
</tbody>
</table>

2.4.21 GenPollText

**class** libqtile.widget.GenPollText(**config**)  
A generic text widget that polls using poll function to get the text

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>func</td>
<td>None</td>
<td>Poll Function</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>update_interval</td>
<td>600</td>
<td>Update interval in seconds, if none, the widget updates whenever it's done.</td>
</tr>
</tbody>
</table>

2.4. Built-in Widgets
2.4.22 GenPollUrl

class libqtile.widget.GenPollUrl(**config)
   A generic text widget that polls an url and parses it using parse function

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>data</td>
<td>None</td>
<td>Post Data</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>headers</td>
<td>{}</td>
<td>Extra Headers</td>
</tr>
<tr>
<td>json</td>
<td>True</td>
<td>Is Json?</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>parse</td>
<td>None</td>
<td>Parse Function</td>
</tr>
<tr>
<td>update_interval</td>
<td>600</td>
<td>Update interval in seconds, if none, the widget updates whenever it's done.</td>
</tr>
<tr>
<td>url</td>
<td>None</td>
<td>Url</td>
</tr>
<tr>
<td>user_agent</td>
<td>'Qtile'</td>
<td>Set the user agent</td>
</tr>
<tr>
<td>xml</td>
<td>False</td>
<td>Is XML?</td>
</tr>
</tbody>
</table>

2.4.23 GmailChecker

class libqtile.widget.GmailChecker(**config)
   A simple gmail checker. If 'status_only_unseen' is True - set 'fmt' for one argument, ex. 'unseen: {0}'

Supported bar orientations: horizontal and vertical
### 2.4.24 GroupBox

**class** `libqtile.widget.GroupBox(**config)**

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>'FFFFFF'</td>
<td>Active group font colour</td>
</tr>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>block_highlight_text_color</td>
<td></td>
<td>Selected group font colour</td>
</tr>
<tr>
<td>borderwidth</td>
<td>3</td>
<td>Current group border width</td>
</tr>
<tr>
<td>center_aligned</td>
<td>True</td>
<td>center-aligned group box</td>
</tr>
<tr>
<td>disable_drag</td>
<td>False</td>
<td>Disable dragging and dropping of group names on widget</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'fffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>hide_unused</td>
<td>False</td>
<td>Hide groups that have no windows and that are not displayed on any screen.</td>
</tr>
<tr>
<td>highlight_color</td>
<td>['000000', '282828']</td>
<td>Active group highlight color when using 'line' highlight method.</td>
</tr>
<tr>
<td>highlight_method</td>
<td>'border'</td>
<td>Method of highlighting ('border', 'block', 'text', or 'line')Uses *_border color settings</td>
</tr>
<tr>
<td>inactive</td>
<td>'404040'</td>
<td>Inactive group font colour</td>
</tr>
<tr>
<td>invert_mouse_wheel</td>
<td>False</td>
<td>Whether to invert mouse wheel group movement</td>
</tr>
<tr>
<td>margin</td>
<td>3</td>
<td>Margin inside the box</td>
</tr>
<tr>
<td>margin_x</td>
<td>None</td>
<td>X Margin. Overrides 'margin' if set</td>
</tr>
</tbody>
</table>

continues on next page
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>margin_y</td>
<td>None</td>
<td>Y Margin. Overrides 'margin' if set</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>other_current_screen_border</td>
<td></td>
<td>Border or line colour for group on other screen when focused.</td>
</tr>
<tr>
<td>other_screen_border</td>
<td>404040</td>
<td>Border or line colour for group on other screen when unfocused.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>padding_x</td>
<td>None</td>
<td>X Padding. Overrides 'padding' if set</td>
</tr>
<tr>
<td>padding_y</td>
<td>None</td>
<td>Y Padding. Overrides 'padding' if set</td>
</tr>
<tr>
<td>rounded</td>
<td>True</td>
<td>To round or not to round box borders</td>
</tr>
<tr>
<td>spacing</td>
<td>None</td>
<td>Spacing between groups (if set to None, will be equal to margin_x)</td>
</tr>
<tr>
<td>this_current_screen_border</td>
<td></td>
<td>Border or line colour for group on this screen when focused.</td>
</tr>
<tr>
<td>this_screen_border</td>
<td>215578</td>
<td>Border or line colour for group on this screen when unfocused.</td>
</tr>
<tr>
<td>urgent_alert_method</td>
<td>'border'</td>
<td>Method for alerting you of WM urgent hints (one of 'border', 'text', 'block', or 'line')</td>
</tr>
<tr>
<td>urgent_border</td>
<td>'FF0000'</td>
<td>Urgent border or line color</td>
</tr>
<tr>
<td>urgent_text</td>
<td>'FF0000'</td>
<td>Urgent group font color</td>
</tr>
<tr>
<td>use_mouse_wheel</td>
<td>True</td>
<td>Whether to use mouse wheel events</td>
</tr>
<tr>
<td>visible_groups</td>
<td>None</td>
<td>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.</td>
</tr>
</tbody>
</table>

#### 2.4.25 HDDBusyGraph

class libqtile.widget.HDDBusyGraph(**config)**

Display HDD busy time graph


Supported bar orientations: horizontal only

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

class libqtile.widget.HDDGraph(**config)
Display HDD free or used space graph

Supported bar orientations: horizontal only

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

2.4.27 IdleRPG

class libqtile.widget.IdleRPG(**config)
A widget for monitoring and displaying IdleRPG stats.

```python
# 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
class `libqtile.widget.Image(length=CALCULATED, **config)`
Display a PNG image on the bar

Supported bar orientations: horizontal and vertical

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

```python
class libqtile.widget.ImapWidget(**config)
```

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>label</td>
<td>'INBOX'</td>
<td>label for display</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mbox</td>
<td>''INBOX''</td>
<td>mailbox to fetch</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>server</td>
<td>None</td>
<td>email server name</td>
</tr>
<tr>
<td>update_interval</td>
<td>600</td>
<td>Update interval in seconds, if none, the widget updates whenever it's done.</td>
</tr>
<tr>
<td>user</td>
<td>None</td>
<td>email username</td>
</tr>
</tbody>
</table>
2.4.30 KeyboardKbdd

**class libqtile.widget.KeyboardKbdd(**config)

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>colours</td>
<td>None</td>
<td>foreground colour for each layout either 'None' or a list of colours. example: ['ffffff', 'E6F0AF'].</td>
</tr>
<tr>
<td>configured_keyboards</td>
<td>['us', 'ir']</td>
<td>your predefined list of keyboard layouts. example: ['us', 'ir', 'es']</td>
</tr>
<tr>
<td>fmt</td>
<td>'{ }'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>update_interval</td>
<td>1</td>
<td>Update interval in seconds.</td>
</tr>
</tbody>
</table>

2.4.31 KeyboardLayout

**class libqtile.widget.KeyboardLayout(**config)

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.

Supported bar orientations: horizontal and vertical
### 2.4.32 KhalCalendar

**class** libqtile.widget.KhalCalendar(**config**)

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>configured_keyboards</td>
<td>['us']</td>
<td>A list of predefined keyboard layouts represented as strings. For example: ['us', 'us colemak', 'es', 'fr'].</td>
</tr>
<tr>
<td>display_map</td>
<td>{}</td>
<td>Custom display of layout. Key should be in format 'layout variant'. For example: {'us': 'us', 'lt sgs': 'sgs', 'ru phonetic': 'ru'}</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>option</td>
<td>None</td>
<td>String of setxkbmap option. Ex., 'compose:menu,grp_led:scroll'</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>update_interval</td>
<td>1</td>
<td>Update time in seconds.</td>
</tr>
<tr>
<td>lookahead</td>
<td>7</td>
<td>Days to look ahead in the calendar</td>
</tr>
<tr>
<td>reminder_color</td>
<td>'FF0000'</td>
<td>Color of calendar entries during reminder time</td>
</tr>
<tr>
<td>remindertime</td>
<td>10</td>
<td>Reminder time in minutes</td>
</tr>
<tr>
<td>update_interval</td>
<td>600</td>
<td>Update interval in seconds, if none, the widget updates whenever it's done.</td>
</tr>
</tbody>
</table>
2.4.33 LaunchBar

class libqtile.widget.LaunchBar(progs=None, width=CALCULATED, **config)
A widget that display icons to launch the associated command

Widget requirements: pyxdg.

Parameters

- **progs** a list of tuples (software_name, command_to_execute, comment). for example:
  - ('thunderbird', 'thunderbird -safe-mode', 'launch thunderbird in safe mode')
  - ('logout', 'qshell:self.qtile.cmd_shutdown()', 'logout from qtile')

Supported bar orientations: horizontal only

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>default_icon</td>
<td>'/usr/share/icons/oxygen/256x256/mimetypes/application-x-executable.png'</td>
<td>Default icon not found</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>2</td>
<td>Padding between icons</td>
</tr>
</tbody>
</table>

2.4.34 Maildir

class libqtile.widget.Maildir(**config)
A simple widget showing the number of new mails in maildir mailboxes

Supported bar orientations: horizontal and vertical
### 2.4.35 Memory

**class** `libqtile.widget.Memory(**config)`

Displays memory/swap usage

MemUsed: Returns memory in use MemTotal: Returns total amount of memory MemFree: Returns amount of memory free MemPercent: Returns memory in use as a percentage Buffers: Returns buffer amount Active: Returns active memory Inactive: Returns inactive memory Shmem: Returns shared memory SwapTotal: Returns total amount of swap SwapFree: Returns amount of swap free SwapUsed: Returns amount of swap in use SwapPercent: Returns swap in use as a percentage

Widget requirements: `psutil`.

Supported bar orientations: horizontal and vertical
### 2.4.36 MemoryGraph

```python
class libqtile.widget.MemoryGraph(**config)
```

Displays a memory usage graph.

**Widget requirements:** `psutil`.

**Supported bar orientations:** horizontal only

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

---

**Markup:**

- **key** default description
- **background** None Widget background color
- **fmt** '{}' How to format the text
- **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.
- **update_interval** 1.0 Update interval for the Memory

---

**Measurements for Memory:**

- **measures_mem**
  - 'G': gigabytes
  - 'M': megabytes
  - 'K': kilobytes
  - 'B': bytes

**Measurements for Swap:**

- 'M': megabytes

---

**Samples:**

- **samples**
  - 100: Count of graph samples.

**Start Position:**

- **start_pos**
  - 'bottom': Drawer starting position ('bottom'/top')
2.4.37 Mirror

**class** libqtile.widget.Mirror(*reflection, **config*)

A widget for showing the same widget content in more than one place, for instance, on bars across multiple screens.

You don't need to use it directly; instead, just instantiate your widget once and hand it in to multiple bars. For instance:

```python
cpu = widget.CPUGraph()
clock = widget.Clock()
screens = [
    Screen(top=bar.Bar([widget.GroupBox(), cpu, clock])),
    Screen(top=bar.Bar([widget.GroupBox(), cpu, clock])),
]
```

Widgets can be passed to more than one bar, so that there don't need to be any duplicates executing the same code all the time, and they'll always be visually identical.

This works for all widgets that use *drawers* (and nothing else) to display their contents. Currently, this is all widgets except for *Systray*.

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
</tbody>
</table>

2.4.38 Moc

**class** libqtile.widget.Moc(**config*)

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
### 2.4.39 `Mpd2`

**class** `libqtile.widget.Mpd2(**config)`

Mpd2 Object.

#### Parameters

**status_format:** format string to display status

*For a full list of values, see:* [MPDClient.status()](https://musicpd.org/doc/protocol/command_reference.html#command_status) and [MPDClient.currentsong()](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 formatting.

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

Default:

```
'{play_status} {idle_message} 
{{repeat}{random}{single}{consume}{updating_db}}'
```

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

Default: "MPD IDLE"

**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.

---

<table>
<thead>
<tr>
<th>Key</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color. default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>noplay_color</td>
<td>'cecece'</td>
<td>Text colour when not playing.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>play_color</td>
<td>'00ff00'</td>
<td>Text colour when playing.</td>
</tr>
<tr>
<td>update_interval</td>
<td>0.5</td>
<td>Update Time in seconds.</td>
</tr>
</tbody>
</table>
Default:

```python
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
<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>color_progress</td>
<td>None</td>
<td>Text color to indicate track progress.</td>
</tr>
<tr>
<td>command</td>
<td>&lt;function default_cmd at 0x7fcf586fab90&gt;</td>
<td>command to be executed by mapped mouse button.</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format_fns</td>
<td>{'all': &lt;function escape at 0x7fcf5d83f8c0&gt;}</td>
<td>Dictionary of format methods</td>
</tr>
<tr>
<td>host</td>
<td>'localhost'</td>
<td>Host of mpd server</td>
</tr>
<tr>
<td>idle_format</td>
<td>'{play_status}{idle_message}{repeat}{random}{single}{consume}{updating_db}'</td>
<td>format for status when mpd has no playlist.</td>
</tr>
<tr>
<td>idle_message</td>
<td>'MPD IDLE'</td>
<td>text to display when mpd is idle.</td>
</tr>
<tr>
<td>idletimeout</td>
<td>5</td>
<td>MPDClient idle command timeout</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>no_connection</td>
<td>'No connection'</td>
<td>Text when mpd is disconnected</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>password</td>
<td>None</td>
<td>Password for auth on mpd server</td>
</tr>
<tr>
<td>play_states</td>
<td>{'pause': '', 'play': '', 'stop': ''}</td>
<td>Play state mapping</td>
</tr>
<tr>
<td>port</td>
<td>6600</td>
<td>Port of mpd server</td>
</tr>
<tr>
<td>prepare_status</td>
<td>{'consume': 'c', 'random': 'z', 'repeat': 'r', 'single': 'l', 'updating_db': 'U'}</td>
<td>characters to show the status of MPD</td>
</tr>
<tr>
<td>space</td>
<td>'-'</td>
<td>Space keeper</td>
</tr>
<tr>
<td>status_format</td>
<td>'{play_status}{artist}/title[{repeat}{random}{single}{consume}{updating_db}]'</td>
<td>format for displayed song info.</td>
</tr>
<tr>
<td>timeout</td>
<td>30</td>
<td>MPDClient timeout</td>
</tr>
<tr>
<td>update_interval</td>
<td>1</td>
<td>Interval of update widget</td>
</tr>
</tbody>
</table>
2.4.40 Mpris2

class libqtile.widget.Mpris2(**config)

An MPRIS 2 widget

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

Widget requirements: dbus-next.

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>name</td>
<td>'audacious'</td>
<td>Name of the MPRIS widget.</td>
</tr>
<tr>
<td>objname</td>
<td>'org.mpris.MediaPlayer2.audacious'</td>
<td>DBUS MPRIS 2 compatible player identifier- Find it out with dbus-monitor - Also see: <a href="http://specifications.freedesktop.org/mpris-spec/latest/#Bus-Name-Policy">http://specifications.freedesktop.org/mpris-spec/latest/#Bus-Name-Policy</a></td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>scroll_chars</td>
<td>30</td>
<td>How many chars at once to display.</td>
</tr>
<tr>
<td>scroll_interval</td>
<td>0.5</td>
<td>Scroll delay interval.</td>
</tr>
<tr>
<td>scroll_wait_intervals</td>
<td>8</td>
<td>Wait x scroll_interval before scrolling/removing text</td>
</tr>
<tr>
<td>stop_pause_text</td>
<td>None</td>
<td>Optional text to display when in the stopped/paused state</td>
</tr>
</tbody>
</table>

2.4.41 Net

class libqtile.widget.Net(**config)

Displays interface down and up speed

Widget requirements: psutil.

Supported bar orientations: horizontal and vertical
<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color. Default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>ffffffff</td>
<td>Foreground color</td>
</tr>
<tr>
<td>format</td>
<td>'{interface}: {down} ↓↑ {up}'</td>
<td>Display format of down/upload/total speed of given interfaces</td>
</tr>
<tr>
<td>interface</td>
<td>None</td>
<td>List of interfaces or single NIC as string to monitor. None to display all active NICs combined</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>prefix</td>
<td>None</td>
<td>Use a specific prefix for the unit of the speed.</td>
</tr>
<tr>
<td>update_interval</td>
<td>1</td>
<td>The update interval.</td>
</tr>
<tr>
<td>use_bits</td>
<td>False</td>
<td>Use bits instead of bytes per second?</td>
</tr>
</tbody>
</table>

### 2.4.42 NetGraph

**class libqtile.widget.NetGraph(**config**

Display a network usage graph.

Widget requirements: psutil.

Supported bar orientations: horizontal only

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

**class libqtile.widget.Notify(width=CALCULATED, **config)**

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>action</td>
<td>True</td>
<td>Enable handling of default action upon right click</td>
</tr>
<tr>
<td>audiofile</td>
<td>None</td>
<td>Audiofile played during notifications</td>
</tr>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>default_timeout</td>
<td>None</td>
<td>Default timeout (seconds) for notifications</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>foreground_low</td>
<td>'dddddd'</td>
<td>Foreground low priority colour</td>
</tr>
<tr>
<td>foreground_urgent</td>
<td>'ff0000'</td>
<td>Foreground urgent priority colour</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>parse_text</td>
<td>None</td>
<td>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</td>
</tr>
</tbody>
</table>

2.4.44 NvidiaSensors

**class libqtile.widget.NvidiaSensors(**config)**

Displays temperature, fan speed and performance level Nvidia GPU.

Supported bar orientations: horizontal and vertical
### 2.4.45 OpenWeather

```python
class libqtile.widget.OpenWeather(**config)
```

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
<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app_key</td>
<td>'7834197c233888258f8cb94ae14ef49'</td>
<td>Open Weather access key. A default is provided, but for prolonged use obtaining your own is suggested: <a href="https://home.openweathermap.org/users/sign_up">https://home.openweathermap.org/users/sign_up</a></td>
</tr>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>cityid</td>
<td>None</td>
<td>City ID. Can be looked up on e.g. : <a href="https://openweathermap.org/find">https://openweathermap.org/find</a>. Takes precedence over location and coordinates. Note that this is not equal to a WOEID.</td>
</tr>
<tr>
<td>coordinates</td>
<td>None</td>
<td>Dictionary containing latitude and longitude. Example: coordinates={&quot;longitude&quot;: &quot;77.22&quot;, &quot;latitude&quot;: &quot;28.67&quot;}</td>
</tr>
<tr>
<td>data</td>
<td>None</td>
<td>Post Data</td>
</tr>
<tr>
<td>dateformat</td>
<td>'%%Y-%%m-%%d'</td>
<td>Format for dates, defaults to ISO. For details see: <a href="https://docs.python.org/3/library/time.html#time.strftime">https://docs.python.org/3/library/time.html#time.strftime</a></td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
</tbody>
</table>
| format     | '{location_city}:
{main_temp}
{units_temperature}
{humidity}%
{weather_details}' | Display format                                                             |
| headers    | {}            | Extra Headers                                                              |
| json       | True          | Is Json?                                                                   |
| language   | 'en'          | Language of response. List of languages supported can be seen at: https://openweathermap.org/current under Multilingual support |
| location   | None          | Name of the city. Country name can be appended like cambridge,NZ. Takes precedence over zip-code. |
| markup     | True          | Whether or not to use pango markup                                          |
| 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                                                             |
| timeformat | '%H:%M'       | Format for times, defaults to ISO. 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 whenever it's done. |
| 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" for other countries. E.g. 12345,NZ. Takes precedence overn coordinates. |
2.4.46 Pomodoro

```python
class libqtile.widget.Pomodoro(**config)
```

Pomodoro technique widget

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>color_active</td>
<td>'00ff00'</td>
<td>Colour then pomodoro is running</td>
</tr>
<tr>
<td>color_break</td>
<td>'ffff00'</td>
<td>Colour then it is break time</td>
</tr>
<tr>
<td>color_inactive</td>
<td>'ff0000'</td>
<td>Colour then pomodoro is inactive</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>length_long_break</td>
<td>15</td>
<td>Length of a long break in minutes</td>
</tr>
<tr>
<td>length_pomodori</td>
<td>25</td>
<td>Length of one pomodori in minutes</td>
</tr>
<tr>
<td>length_short_break</td>
<td>5</td>
<td>Length of a short break in minutes</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions.</td>
</tr>
<tr>
<td>notification_on</td>
<td>True</td>
<td>Turn notifications on</td>
</tr>
<tr>
<td>num_pomodori</td>
<td>4</td>
<td>Number of pomodori to do in a cycle</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>prefix_active</td>
<td>''</td>
<td>Prefix then app is active</td>
</tr>
<tr>
<td>prefix_break</td>
<td>'B'</td>
<td>Prefix during short break</td>
</tr>
<tr>
<td>prefix_inactive</td>
<td>'POMODORO'</td>
<td>Prefix when app is inactive</td>
</tr>
<tr>
<td>prefix_long_break</td>
<td>'LB'</td>
<td>Prefix during long break</td>
</tr>
<tr>
<td>prefix_paused</td>
<td>'PAUSE'</td>
<td>Prefix during pause</td>
</tr>
<tr>
<td>update_interval</td>
<td>1</td>
<td>Update interval in seconds, if none, the widget updates whenever the event loop is idle.</td>
</tr>
</tbody>
</table>

2.4.47 Prompt

```python
class libqtile.widget.Prompt(name='prompt', **config)
```

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
<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>bell_style</td>
<td>'audible'</td>
<td>Alert at the begin/end of the command history. Possible values: 'audible' (X11 only), 'visual' and None.</td>
</tr>
<tr>
<td>cursor</td>
<td>True</td>
<td>Show a cursor</td>
</tr>
<tr>
<td>cursor_color</td>
<td>'bef098'</td>
<td>Color for the cursor and text over it.</td>
</tr>
<tr>
<td>cursorblink</td>
<td>0.5</td>
<td>Cursor blink rate. 0 to disable.</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>ignore_dups_history</td>
<td>False</td>
<td>Don't store duplicates in history</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>max_history</td>
<td>100</td>
<td>Commands to keep in history. 0 for no limit.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>prompt</td>
<td>'{prompt}: '</td>
<td>Text displayed at the prompt</td>
</tr>
<tr>
<td>record_history</td>
<td>True</td>
<td>Keep a record of executed commands</td>
</tr>
<tr>
<td>visual_bell_color</td>
<td>'ff0000'</td>
<td>Color for the visual bell (changes prompt background).</td>
</tr>
<tr>
<td>visual_bell_time</td>
<td>0.2</td>
<td>Visual bell duration (in seconds).</td>
</tr>
</tbody>
</table>

### 2.4.48 PulseVolume

```python
class libqtile.widget.PulseVolume(**config)
```

Supported bar orientations: horizontal only
<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>cardid</td>
<td>None</td>
<td>Card Id</td>
</tr>
<tr>
<td>channel</td>
<td>'Master'</td>
<td>Channel</td>
</tr>
<tr>
<td>device</td>
<td>'default'</td>
<td>Device Name</td>
</tr>
<tr>
<td>emoji</td>
<td>False</td>
<td>Use emoji to display volume states, only if theme_path is not set. The specified font needs to contain the correct unicode characters.</td>
</tr>
<tr>
<td>fmt</td>
<td>'{};'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>get_volume_command</td>
<td>None</td>
<td>Command to get the current volume</td>
</tr>
<tr>
<td>limit_max_volume</td>
<td>False</td>
<td>Limit maximum volume to 100%</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>mute_command</td>
<td>None</td>
<td>Mute command</td>
</tr>
<tr>
<td>padding</td>
<td>3</td>
<td>Padding left and right. Calculated if None.</td>
</tr>
<tr>
<td>step</td>
<td>2</td>
<td>Volume change for up and down commands in percentage. Only used if volume_up_command and volume_down_command are not set.</td>
</tr>
<tr>
<td>theme_path</td>
<td>None</td>
<td>Path of the icons</td>
</tr>
<tr>
<td>update_interval</td>
<td>0.2</td>
<td>Update time in seconds.</td>
</tr>
<tr>
<td>volume_app</td>
<td>None</td>
<td>App to control volume</td>
</tr>
<tr>
<td>volume_down_command</td>
<td>None</td>
<td>Volume down command</td>
</tr>
<tr>
<td>volume_up_command</td>
<td>None</td>
<td>Volume up command</td>
</tr>
</tbody>
</table>

### 2.4.49 QuickExit

**class** `libqtile.widget.QuickExit(widget=CALCULATED, **config)`

A button of exiting the running qtile easily. When clicked this button, a countdown start. If the button pushed with in the countdown again, the qtile shutdown.

Supported bar orientations: horizontal and vertical
key | default | description
---|---|---
background | None | Widget background color

countdown_format | '{ {} seconds }' | This text is showed when counting down.

countdown_start | 5 | Time to accept the second pushing.

default_text | '{ shutdown }' | A text displayed as a button

fmt | '{ }' | How to format the text

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 | 2 | Padding. Calculated if None.

timer_interval | 1 | A countdown interval.

### 2.4.50 Sep

**class** libqtile.widget.Sep(**config**)
A visible widget separator

Supported bar orientations: horizontal and vertical

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

### 2.4.51 She

**class** libqtile.widget.She(**config**)

A 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
2.4.52 Spacer

```python
class libqtile.widget.Spacer(length=STRETCH, **config)
```

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Acceps functions and lazy calls.</td>
</tr>
</tbody>
</table>

2.4.53 StatusNotifier

```python
class libqtile.widget.StatusNotifier(**config)
```

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 xdg module to support retrieving icons from the selected theme.

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>icon_size</td>
<td>16</td>
<td>Icon width</td>
</tr>
<tr>
<td>icon_theme</td>
<td>None</td>
<td>Name of theme to use for app icons</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>3</td>
<td>Padding between icons</td>
</tr>
</tbody>
</table>

### 2.4.54 StockTicker

class libqtile.widget.StockTicker(**config)

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.

```python
# 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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>data</td>
<td>None</td>
<td>Post Data</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground color</td>
</tr>
<tr>
<td>function</td>
<td>'TIME_SERIES_INTRADAY'</td>
<td>Default API function to query</td>
</tr>
<tr>
<td>headers</td>
<td>{}</td>
<td>Extra Headers</td>
</tr>
<tr>
<td>interval</td>
<td>'1min'</td>
<td>The default latency to query</td>
</tr>
<tr>
<td>json</td>
<td>True</td>
<td>Is Json?</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>parse</td>
<td>None</td>
<td>Parse Function</td>
</tr>
<tr>
<td>update_interval</td>
<td>600</td>
<td>Update interval in seconds, if none, the widget updates whenever it's done.</td>
</tr>
<tr>
<td>url</td>
<td>None</td>
<td>Url</td>
</tr>
<tr>
<td>user_agent</td>
<td>'Qtile'</td>
<td>Set the user agent</td>
</tr>
<tr>
<td>xml</td>
<td>False</td>
<td>Is XML?</td>
</tr>
</tbody>
</table>
2.4.55 SwapGraph

```python
class libqtile.widget.SwapGraph(**config)
```
Display a swap info graph.

Widget requirements: psutil.

Supported bar orientations: horizontal only

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

2.4.56 Systray

```python
class libqtile.widget.Systray(**config)
```
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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>icon_size</td>
<td>20</td>
<td>Icon width</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>5</td>
<td>Padding between icons</td>
</tr>
</tbody>
</table>
2.4.57 TaskList

class libqtile.widget.TaskList(**config)

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.

Supported bar orientations: horizontal only

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>border</td>
<td>'215578'</td>
<td>Border colour</td>
</tr>
<tr>
<td>borderwidth</td>
<td>2</td>
<td>Current group border width</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>highlight_method</td>
<td>'border'</td>
<td>Method of highlighting (one of 'border' or 'block') Uses _border color settings</td>
</tr>
<tr>
<td>icon_size</td>
<td>None</td>
<td>Icon size. (Calculated if set to None. Icons are hidden if set to 0.)</td>
</tr>
<tr>
<td>margin</td>
<td>3</td>
<td>Margin inside the box</td>
</tr>
<tr>
<td>margin_x</td>
<td>None</td>
<td>X Margin. Overrides 'margin' if set</td>
</tr>
<tr>
<td>margin_y</td>
<td>None</td>
<td>Y Margin. Overrides 'margin' if set</td>
</tr>
<tr>
<td>markup Floating</td>
<td>None</td>
<td>Text markup of the floating window state. Supports pangomarkup with markup=True.e.g., &quot;{}&quot; or &quot;&lt;span underline=&quot;low&quot;&gt;{}&lt;/span&gt;&quot;</td>
</tr>
<tr>
<td>markup_focused</td>
<td>None</td>
<td>Text markup of the focused window state. Supports pangomarkup with markup=True.e.g., &quot;{}&quot; or &quot;&lt;span underline=&quot;low&quot;&gt;{}&lt;/span&gt;&quot;</td>
</tr>
<tr>
<td>markup_maximized</td>
<td>None</td>
<td>Text markup of the maximized window state. Supports pangomarkup with markup=True.e.g., &quot;{}&quot; or &quot;&lt;span underline=&quot;low&quot;&gt;{}&lt;/span&gt;&quot;</td>
</tr>
<tr>
<td>markup_minimized</td>
<td>None</td>
<td>Text markup of the minimized window state. Supports pangomarkup with markup=True.e.g., &quot;{}&quot; or &quot;&lt;span underline=&quot;low&quot;&gt;{}&lt;/span&gt;&quot;</td>
</tr>
<tr>
<td>markup_normal</td>
<td>None</td>
<td>Text markup of the normal window state. Supports pangomarkup with markup=True.e.g., &quot;{}&quot; or &quot;&lt;span underline=&quot;low&quot;&gt;{}&lt;/span&gt;&quot;</td>
</tr>
<tr>
<td>max_title_width</td>
<td>None</td>
<td>Max size in pixels of task title.(if set to None, as much as available.)</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>3</td>
<td>Padding inside the box</td>
</tr>
<tr>
<td>padding_x</td>
<td>None</td>
<td>X Padding. Overrides 'padding' if set</td>
</tr>
<tr>
<td>padding_y</td>
<td>None</td>
<td>Y Padding. Overrides 'padding' if set</td>
</tr>
<tr>
<td>parse_text</td>
<td>None</td>
<td>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 [&quot; - Chromium&quot;, &quot; - Firefox&quot;]: text = text.replace(string, &quot;) return text then set option parse_text=my_func</td>
</tr>
<tr>
<td>rounded</td>
<td>True</td>
<td>To round or not to round borders</td>
</tr>
<tr>
<td>spacing</td>
<td>None</td>
<td>Spacing between tasks.(if set to None, will be equal to margin_x)</td>
</tr>
</tbody>
</table>

continues on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>title_width_method</td>
<td>None</td>
<td>Method to compute the width of task title.  (None, uniform). Defaults to None, the normal behaviour.</td>
</tr>
<tr>
<td>txt_floating</td>
<td>'V'</td>
<td>Text representation of the floating window state. e.g., ”V” or ””.</td>
</tr>
<tr>
<td>txt_maximized</td>
<td>'[]'</td>
<td>Text representation of the maximized window state. e.g., ”[]” or ””</td>
</tr>
<tr>
<td>txt_minimized</td>
<td>'_'</td>
<td>Text representation of the minimized window state. e.g., ”_” or ””</td>
</tr>
<tr>
<td>unfocused_border</td>
<td>None</td>
<td>Border color for unfocused windows. Affects only highlight_method 'border' and 'block'. Defaults to None, which means no special color.</td>
</tr>
<tr>
<td>urgent_alert_method</td>
<td>'border'</td>
<td>Method for alerting you of WM urgent hints (one of 'border' or 'text')</td>
</tr>
<tr>
<td>urgent_border</td>
<td>'FF0000'</td>
<td>Urgent border color</td>
</tr>
</tbody>
</table>

### 2.4.58 TextBox

**class** `libqtile.widget.TextBox`(*text='*', width=CALCULATED, **config*)

A flexible textbox that can be updated from bound keys, scripts, and qshell.

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Text font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font pixel size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'#ffffff'</td>
<td>Foreground colour.</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding left and right. Calculated if None.</td>
</tr>
</tbody>
</table>

### 2.4.59 ThermalSensor

**class** `libqtile.widget.ThermalSensor(**config)`

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
### 2.4.60 ThermalZone

**class** `libqtile.widget.ThermalZone(**config)`

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

---

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>foreground_alert</td>
<td>'ff0000'</td>
<td>Foreground colour alert</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>metric</td>
<td>True</td>
<td>True to use metric/C, False to use imperial/F</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Acceps functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>show_tag</td>
<td>False</td>
<td>Show tag sensor</td>
</tr>
<tr>
<td>tag_sensor</td>
<td>None</td>
<td>Tag of the temperature sensor. For example: &quot;temp1&quot; or &quot;Core 0&quot;</td>
</tr>
<tr>
<td>threshold</td>
<td>70</td>
<td>If the current temperature value is above, then change to foreground_alert colour</td>
</tr>
<tr>
<td>update_interval</td>
<td>2</td>
<td>Update interval in seconds</td>
</tr>
<tr>
<td>key</td>
<td>default</td>
<td>description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>crit</td>
<td>70</td>
<td>Critical temperature level</td>
</tr>
<tr>
<td>fgcolor_crit</td>
<td>'ff0000'</td>
<td>Font color on critical values</td>
</tr>
<tr>
<td>fgcolor_high</td>
<td>'ffaa00'</td>
<td>Font color on high values</td>
</tr>
<tr>
<td>fgcolor_normal</td>
<td>'ffffff'</td>
<td>Font color on normal values</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'{temp}°C'</td>
<td>Display format</td>
</tr>
<tr>
<td>format_crit</td>
<td>'{temp}°C CRIT!'</td>
<td>Critical display format</td>
</tr>
<tr>
<td>hidden</td>
<td>False</td>
<td>Set True to only show if critical value reached</td>
</tr>
<tr>
<td>high</td>
<td>50</td>
<td>High temperature level</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>update_interval</td>
<td>2.0</td>
<td>Update interval</td>
</tr>
<tr>
<td>zone</td>
<td>'/sys/class/thermal/thermal_zone0/temp'</td>
<td>Thermal zone</td>
</tr>
</tbody>
</table>

### 2.4.61 Volume

```python
class libqtile.widget.Volume(**config)

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
<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>cardid</td>
<td>None</td>
<td>Card Id</td>
</tr>
<tr>
<td>channel</td>
<td>'Master'</td>
<td>Channel</td>
</tr>
<tr>
<td>device</td>
<td>'default'</td>
<td>Device Name</td>
</tr>
<tr>
<td>emoji</td>
<td>False</td>
<td>Use emoji to display volume states, only if <code>theme_path</code> is not set. The specified font needs to contain the correct unicode characters.</td>
</tr>
<tr>
<td>fmt</td>
<td>'{ }'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'#ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>get_volume_command</td>
<td>None</td>
<td>Command to get the current volume</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>mute_command</td>
<td>None</td>
<td>Mute command</td>
</tr>
<tr>
<td>padding</td>
<td>3</td>
<td>Padding left and right. Calculated if None.</td>
</tr>
<tr>
<td>step</td>
<td>2</td>
<td>Volume change for up and down commands in percentage. Only used if <code>volume_up_command</code> and <code>volume_down_command</code> are not set.</td>
</tr>
<tr>
<td>theme_path</td>
<td>None</td>
<td>Path of the icons</td>
</tr>
<tr>
<td>update_interval</td>
<td>0.2</td>
<td>Update time in seconds.</td>
</tr>
<tr>
<td>volume_app</td>
<td>None</td>
<td>App to control volume</td>
</tr>
<tr>
<td>volume_down_command</td>
<td>None</td>
<td>Volume down command</td>
</tr>
<tr>
<td>volume_up_command</td>
<td>None</td>
<td>Volume up command</td>
</tr>
</tbody>
</table>

### 2.4.62 Wallpaper

```python
class libqtile.widget.Wallpaper(**config)

Supported bar orientations: horizontal and vertical
```
### 2.4.63 WidgetBox

**class** `libqtile.widget.WidgetBox(widgets: Optional[list] = None, **config)`

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:

```python
widget.WidgetBox(widgets=[
    widget.TextBox(text="This widget is in the box"),
    widget.Memory()
],

widget.WidgetBox(widgets=[
    widget.TextBox(text="This widget is in the box"),
    widget.Memory()
],

Supported bar orientations: horizontal only

### 2.4. Built-in Widgets

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>directory</td>
<td>'~/Pictures/wallpapers/'</td>
<td>Wallpaper Directory</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>label</td>
<td>None</td>
<td>Use a fixed label instead of image name.</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>option</td>
<td>'fill'</td>
<td>How to fit the wallpaper when wallpaper_command is None. None, 'fill' or 'stretch'.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>random_selection</td>
<td>False</td>
<td>If set, use random initial wallpaper and randomly cycle through the wallpapers.</td>
</tr>
<tr>
<td>wallpaper</td>
<td>None</td>
<td>Wallpaper</td>
</tr>
<tr>
<td>wallpaper_command</td>
<td>['feh', '--bg-fill']</td>
<td>Wallpaper command. If None, the wallpaper will be painted without the use of a helper.</td>
</tr>
</tbody>
</table>
### 2.4.64 WindowCount

**class** `libqtile.widget.WindowCount` *(text='', width=``CALCULATED``*, **config)*

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Text font</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font pixel size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'#ffffff'</td>
<td>Foreground colour.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>text_closed</td>
<td>'['</td>
<td>Text when box is closed</td>
</tr>
<tr>
<td>text_open</td>
<td>'['</td>
<td>Text when box is open</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding left and right. Calculated if None.</td>
</tr>
<tr>
<td>show_zero</td>
<td>False</td>
<td>Show window count when no windows</td>
</tr>
<tr>
<td>text_format</td>
<td>'{num}'</td>
<td>Format for message</td>
</tr>
</tbody>
</table>

### 2.4.65 WindowName

**class** `libqtile.widget.WindowName` *(width=``STRETCH``*, **config)*

Displays the name of the window that currently has focus.

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Text font</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font pixel size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'#ffffff'</td>
<td>Foreground colour.</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding left and right. Calculated if None.</td>
</tr>
<tr>
<td>show_zero</td>
<td>False</td>
<td>Show window count when no windows</td>
</tr>
<tr>
<td>text_format</td>
<td>'{num}'</td>
<td>Format for message</td>
</tr>
</tbody>
</table>
### 2.4.66 WindowTabs

**class** libqtile.widget.WindowTabs(**config**)

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

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>empty_group_string</td>
<td>' '</td>
<td>string to display when no windows are focused on current group</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None(no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>for_current_screen</td>
<td>False</td>
<td>instead of this bars screen use currently active screen</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'{state}{name}'</td>
<td>format of the text</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>parse_text</td>
<td>None</td>
<td>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 [&quot; - Chromium&quot;, &quot; - Firefox&quot;]: text = text.replace(string, &quot;) return text then set option parse_text=my_func</td>
</tr>
<tr>
<td>selected</td>
<td>('&lt;b&gt;', ' '&lt;/b&gt;')</td>
<td>Selected task indicator</td>
</tr>
<tr>
<td>separator</td>
<td>'</td>
<td>'</td>
</tr>
</tbody>
</table>

---

2.4. Built-in Widgets
2.4.67 Wlan

class libqtile.widget.Wlan(**config)
Displays Wifi SSID and quality.

Widget requirements: iwlib.

Supported bar orientations: horizontal only

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>disconnected_message</td>
<td>'Disconnected'</td>
<td>String to show when the wlan is disconnected.</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontshadow</td>
<td>None</td>
<td>Font shadow color, default is None (no shadow)</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'{essid} {quality}/70'</td>
<td>Display format. For percents you can use &quot;[essid] {percentage:2.0%}&quot;</td>
</tr>
<tr>
<td>interface</td>
<td>'wlan0'</td>
<td>The interface to monitor</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>update_interval</td>
<td>1</td>
<td>The update interval</td>
</tr>
</tbody>
</table>

2.4.68 Wttr

class libqtile.widget.Wttr(**config)
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

```python
location={
    'Minsk': 'Minsk',
    '64.127146,-21.873472': 'Reykjavik',
}
```

Cities will change randomly every update.

Supported bar orientations: horizontal and vertical

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>None</td>
<td>Widget background color</td>
</tr>
<tr>
<td>data</td>
<td>None</td>
<td>Post Data</td>
</tr>
<tr>
<td>fmt</td>
<td>'{}'</td>
<td>How to format the text</td>
</tr>
<tr>
<td>font</td>
<td>'sans'</td>
<td>Default font</td>
</tr>
<tr>
<td>fontsize</td>
<td>None</td>
<td>Font size. Calculated if None.</td>
</tr>
<tr>
<td>foreground</td>
<td>'ffffff'</td>
<td>Foreground colour</td>
</tr>
<tr>
<td>format</td>
<td>'3'</td>
<td>Display text format. Choose presets in range 1-4 (Ex. &quot;1&quot;) or build your own custom output format, use the special %-notation. See <a href="https://github.com/chubin/wttr.in#one-line-output">https://github.com/chubin/wttr.in#one-line-output</a></td>
</tr>
<tr>
<td>headers</td>
<td>{}</td>
<td>Extra Headers</td>
</tr>
<tr>
<td>json</td>
<td>False</td>
<td>Is Json?</td>
</tr>
<tr>
<td>lang</td>
<td>'en'</td>
<td>Display text language. List of supported languages <a href="https://wttr.in/:translation">https://wttr.in/:translation</a></td>
</tr>
<tr>
<td>location</td>
<td>None</td>
<td>Dictionary. Key is a city or place name, or GPS coordinates. Value is a display name.</td>
</tr>
<tr>
<td>markup</td>
<td>True</td>
<td>Whether or not to use pango markup</td>
</tr>
<tr>
<td>max_chars</td>
<td>0</td>
<td>Maximum number of characters to display in widget.</td>
</tr>
<tr>
<td>mouse_callbacks</td>
<td>{}</td>
<td>Dict of mouse button press callback functions. Accepts functions and lazy calls.</td>
</tr>
<tr>
<td>padding</td>
<td>None</td>
<td>Padding. Calculated if None.</td>
</tr>
<tr>
<td>parse</td>
<td>None</td>
<td>Parse Function</td>
</tr>
<tr>
<td>units</td>
<td>'m'</td>
<td>'m' - metric, 'M' - show wind speed in m/s, 'u' - United States units</td>
</tr>
<tr>
<td>update_interval</td>
<td>600</td>
<td>Update interval in seconds. Recommendation: if you want to display multiple locations alternately, maybe set a smaller interval, ex. 30.</td>
</tr>
<tr>
<td>url</td>
<td>None</td>
<td>Url</td>
</tr>
<tr>
<td>user_agent</td>
<td>'Qtile'</td>
<td>Set the user agent</td>
</tr>
<tr>
<td>xml</td>
<td>False</td>
<td>Is XML?</td>
</tr>
</tbody>
</table>
2.5 Default Config File

```python
# 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
# 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"),
```

(continues on next page)
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"], "i", lazy.layout.grow_down(), desc="Grow window down"),
Key([mod, "control"], "j", 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, "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"),
]

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

for i in groups:
    keys.extend(
        [
            # mod1 + letter of group = switch to group
            Key(
                [mod],
                i.name,
                lazy.group[i.name].toscreen(),
                desc="Switch to group {}".format(i.name),
            ),
            # mod1 + shift + letter of group = 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.
            # # mod1 + shift + letter of group = 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(),
]

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"),
             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
    ),
]
# 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()),
    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
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

# 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 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"
3.1 Scripting

3.1.1 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 Commands API section of this manual. Full command documentation is available through the Qtile Shell.

3.1.2 Example

Below is a very minimal example script that inspects the current Qtile instance, and returns the integer offset of the current screen.

```python
from libqtile.command.client import InteractiveCommandClient
c = InteractiveCommandClient()
print(c.screen.info()['index'])
```

3.2 Commands API

Qtile's command API is based on a graph of objects, where each object has a set of associated commands. The graph and object commands are used in a number of different places:

- Commands can be bound to keys in the Qtile configuration file.
- Commands can be called through qtile shell, the Qtile shell.
- The shell can also be hooked into a Jupyter kernel called iqshell.
- Commands can be called from a script to interact with Qtile from Python.

If the explanation below seems a bit complex, please take a moment to explore the API using the qtile shell command shell. Command lists and detailed documentation can be accessed from its built-in help command.
3.2.1 Introduction: Object Graph

The objects in Qtile's object graph come in seven flavours, matching the seven basic components of the window manager: *layouts*, *windows*, *groups*, *bars*, *widgets*, *screens*, and a special *root* node. Objects are addressed by a path specification that starts at the root, and follows the edges of 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:

```python
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:

```python
from libqtile.command.client import CommandClient
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.to_screen()` method):

```
c.group["b"].to_screen(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.

### 3.2.2 Keys

The key specifier for the various object types are as follows:
### 3.2.3 Digging Deeper: Command Objects

If you just want to script your Qtile window manager the above information, in addition to the documentation on the various scripting 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.

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 named `cmd_*`. When writing custom layouts, widgets, or any other object, you can add your own custom `cmd_*` functions and 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.
3.2.4 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.

3.2.5 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() command against the addressed screen object. Additional functionality can be used to check that a given traversal resolves to actual objects 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 associated 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.

3.2.6 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:

```python
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.

### 3.3 Scripting Commands

Here is documented some of the commands available on objects in the command tree when running `qtile shell` or scripting commands to `qtile`. Note that this is an incomplete list, some objects, such as `layouts` and `widgets`, may implement their own set of commands beyond those given here.

#### 3.3.1 Qtile

```python
    This object is the root of the command graph
    
    cmd_add_rule(match_args: Dict[str, Any], rule_args: Dict[str, Any], min_priority: bool = False)
    Add a dgroup rule, returns rule_id needed to remove it

    Parameters
        match_args  config.Match arguments
        rule_args   config.Rule arguments
        min_priority If the rule is added with minimum priority (last) (default: False)

    cmd_addgroup(group: str, label: Optional[str] = None, layout: Optional[str] = None, layouts:
        Optional[List[Layout]] = None) → bool
    Add a group with the given name

    cmd_commands() → List[str]
    Returns a list of possible commands for this object
    Used by __qsh__ for command completion and online help

    cmd_critical() → None
    Set log level to CRITICAL
```
cmd_debug() → None
Set log level to DEBUG

cmd_delgroup(group: str) → None
Delete a group with the given name

cmd_display_kb(*args) → str
Display table of key bindings

cmd_doc(name) → str
Returns the documentation for a specified command name
Used by __qsh__ to provide online help.

cmd_error() → None
Set log level to ERROR

cmd_eval(code: str) → Tuple[bool, Optional[str]]
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.

cmd_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")

cmd_function(function, *args, **kwargs) → None
Call a function with current object as argument

cmd_get_state() → str
Get pickled state for restarting qtile

cmd_get_test_data() → Any
Returns any content arbitrarily set in the self.test_data attribute. Useful in tests.

cmd_groups() → Dict[str, Dict[str, Any]]
Return a dictionary containing information for all groups

Examples

groups()

cmd_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")

cmd_info() → None
Set log level to INFO

cmd_internal_windows() → List[Dict[str, Any]]
Return info for each internal window (bars, for example)

cmd_items(name) → Tuple[bool, Optional[List[Union[str, int]]]]
Returns a list of contained items for the specified name
Used by `__qsh__` to allow navigation of the object graph.

```python
cmd_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")

```python
cmd_list_widgets() → List[str]
```
List of all addressible widget names

```python
cmd_loglevel() → int
```

```python
cmd_loglevelname() → str
```

```python
cmd_next_layout(name: Optional[str] = None) → None
```
Switch to the next layout.

**Parameters**

name Group name. If not specified, the current group is assumed

```python
cmd_next_screen() → None
```
Move to next screen

```python
cmd_next_urgent() → None
```
Focus next window with urgent hint

```python
cmd_pause() → None
```
Drops into pdb

```python
cmd_prev_layout(name: Optional[str] = None) → None
```
Switch to the previous layout.

**Parameters**

name Group name. If not specified, the current group is assumed

```python
cmd_prev_screen() → None
```
Move to the previous screen

```python
cmd_qtile_info() → Dict
```
Returns a dictionary of info on the Qtile instance

```python
cmd_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")

```python
cmd_reconfigure_screens(ev: Any = None) → 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 ev kwarg is ignored; it is here in case this function is hooked directly to screen_change.
cmd_reload_config() → None

Reload the configuration file.

Can also be triggered by sending Qtile a SIGUSR1 signal.

cmd_remove_rule(rule_id: int) → None

Remove a dgroup rule by rule_id

cmd_restart() → None

Restart Qtile.

Can also be triggered by sending Qtile a SIGUSR2 signal.

cmd_run_extension(extension: libqtile.extension.base._Extension) → None

Run extensions

cmd_screens() → List[Dict[str, Any]]

Return a list of dictionaries providing information on all screens

cmd_shutdown() → None

Quit Qtile

cmd_simulate_keypress(modifiers, key) → None

Simulates a keypress on the focused window.

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")

**cmd.spawn**

(cmd: Union[str, List[str]], shell: bool = False) → int

Run cmd, in a shell or not (default).

cmd may be a string or a list (similar to subprocess.Popen).

Examples

spawn("firefox")

spawn(["xterm", ".T", "Temporary terminal"])
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.

```python
cmd_status() → Literal['OK']
```
Return "OK" if Qtile is running

```python
cmd_switch_groups(namea: str, nameb: str) → None
```
Switch position of two groups by name

```python
cmd_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")

```python
cmd_sync() → None
```
Sync the backend's event queue. Should only be used for development.

```python
cmd_to_layout_index(index: str, name: Optional[str] = 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.

```python
cmd_to_screen(n: int) → None
```
Warp focus to screen n, where n is a 0-based screen number

### Examples

to_screen(0)

```python
cmd_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")

```python
cmd_tracemalloc_dump() → Tuple[bool, str]
```
Dump tracemalloc snapshot

```python
cmd_tracemalloc_toggle() → None
```
Toggle tracemalloc status

Running tracemalloc is required for `qtile top`

```python
cmd_ungrab_all_chords() → None
```
Leave all chord modes and grab the root bindings

```python
cmd_ungrab_chord() → None
```
Leave a chord mode

```python
cmd_validate_config() → None
```
Set log level to WARNING
cmd_windows() \to \text{List[Dict[str, Any]]}

Return info for each client window

### 3.3.2 Bar

class \texttt{libqtile.bar.Bar}(\texttt{widgets, size, **config})

A bar, which can contain widgets

**Parameters**

- \texttt{widgets} A list of widget objects.
- \texttt{size} The “thickness” of the bar, i.e. the height of a horizontal bar, or the width of a vertical bar.

<table>
<thead>
<tr>
<th>key</th>
<th>default</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>background</td>
<td>'#000000'</td>
<td>Background colour.</td>
</tr>
<tr>
<td>border_color</td>
<td>'#000000'</td>
<td>Border colour as str or list of str [N E S W]</td>
</tr>
<tr>
<td>border_width</td>
<td>0</td>
<td>Width of border as int of list of ints [N E S W]</td>
</tr>
<tr>
<td>margin</td>
<td>0</td>
<td>Space around bar as int or list of ints [N E S W].</td>
</tr>
<tr>
<td>opacity</td>
<td>1</td>
<td>Bar window opacity.</td>
</tr>
</tbody>
</table>

cmd_commands() \to \text{List[str]}

Returns a list of possible commands for this object

Used by \texttt{__qsh__} for command completion and online help

**cmd_doc**(\texttt{name}) \to \text{str}

Returns the documentation for a specified command name

Used by \texttt{__qsh__} to provide online help.

**cmd_eval**(\texttt{code: str}) \to \text{Tuple[bool, Optional[str]]}

Evaluates code in the same context as this function

Return value is tuple \texttt{(success, result)}, success being a boolean and result being a string representing the return value of eval, or None if exec was used instead.

**cmd_fake_button_press**(\texttt{screen, position, x, y, button=1})

Fake a mouse-button-press on the bar. Co-ordinates are relative to the top-left corner of the bar.

:screen The integer screen offset :position One of "top", "bottom", "left", or "right"

**cmd_function**(\texttt{function, *args, **kwargs}) \to \text{None}

Call a function with current object as argument

**cmd_info**()

Info for this object.

**cmd_items**(\texttt{name}) \to \text{Tuple[bool, Optional[List[Union[str, int]]]]}

Returns a list of contained items for the specified name

Used by \texttt{__qsh__} to allow navigation of the object graph.
3.3.3 Group

class libqtile.config.Group(name: str, matches: Optional[List[libqtile.config.Match]] = None,
exclusive=False, spawn: Optional[Union[str, List[str]]] = None, layout:
Optional[str] = None, layouts: Optional[List] = None, persist=True, init=True,
layout_opts=None, screen_affinity=None, position=922372036854775807,
label: Optional[str] = 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: string the name of this group
matches: default `None` list of Match objects whose windows will be assigned to this group
exclusive: boolean when other apps are started in this group, should we allow them here or not?
spawn: string or list of strings this will be exec() d when the group is created, you can pass either a program name or a list of programs to exec()
layout: string the name of default layout for this group (e.g. 'max' or 'stack'). This is the name specified for a particular layout in config.py or if not defined it defaults in general the class name in all lower case.
layers: list the group layouts list overriding global layouts. Use this to define a separate list of layouts for this particular group.
persist: boolean should this group stay alive with no member windows?
init: boolean is this group alive when Qtile starts?
position int group position
label: string 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.

3.3.4 Screen

class libqtile.config.Screen(top: Optional[Union[libqtile.bar.Bar, libqtile.bar.Gap]] = None, bottom:
Optional[Union[libqtile.bar.Bar, libqtile.bar.Gap]] = None, left:
Optional[Union[libqtile.bar.Bar, libqtile.bar.Gap]] = None, right:
Optional[Union[libqtile.bar.Bar, libqtile.bar.Gap]] = None, wallpaper:
Optional[str] = None, wallpaper_mode: Optional[str] = None, x:
Optional[int] = None, y: Optional[int] = None, width: Optional[int] = None,
height: Optional[int] = None)

A physical screen, and its associated paraphernalia.

Define a screen with a given set of Bars of a specific geometry. Note that Bar objects can only be placed at the top or the bottom of the screen (Bar.Gap objects can be placed anywhere). 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 screen's 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.
cmd_commands() → List[str]
Returns a list of possible commands for this object
Used by __qsh__ for command completion and online help

abstract cmd_disable_floating() → None
Tile the window.

abstract cmd_disable_fullscreen() → None
Un-fullscreen the window

3.3.5 Window

class libqtile.backend.base.Window
A regular Window belonging to a client.
Abstract methods are required to be defined as part of a specific backend's implementation. Non-abstract methods have default implementations here to be shared across backends.

abstract cmd_bring_to_front() → None
Bring the window to the front

abstract cmd_disable_floating() → None
Tile the window.
\texttt{cmd_doc(name) \rightarrow str}

Returns the documentation for a specified command name

Used by \_\_qsh\_\_ to provide online help.

\texttt{cmd_down_opacity() \rightarrow None}

Decrease the window’s opacity by 10%.

\textbf{abstract cmd_enable_floating() \rightarrow None}

Float the window.

\textbf{abstract cmd_enable_fullscreen() \rightarrow None}

Fullscreen the window

\textbf{cmd_eval(code: str) \rightarrow Tuple[bool, Optional[str]]}

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.

\textbf{abstract cmd_focus(warp: bool = True) \rightarrow None}

Focuses the window.

\textbf{cmd_function(function, *args, **kwargs) \rightarrow None}

Call a function with current object as argument

\textbf{abstract cmd_get_position() \rightarrow Tuple[int, int]}

Get the \((x, y)\) of the window

\textbf{abstract cmd_get_size() \rightarrow Tuple[int, int]}

Get the \((width, height)\) of the window

\textbf{cmd_info() \rightarrow Dict}

Return a dictionary of info.

\textbf{cmd_items(name) \rightarrow Tuple[bool, Optional[List[Union[str, int]]]]}

Returns a list of contained items for the specified name

Used by \_\_qsh\_\_ to allow navigation of the object graph.

\textbf{abstract cmd_kill() \rightarrow None}

Kill the window. Try to be polite.

\textbf{cmd_match(*args, **kwargs) \rightarrow bool}

\textbf{abstract cmd_move_floating(dx: int, dy: int) \rightarrow None}

Move window by \(dx\) and \(dy\)

\textbf{cmd_opacity(opacity: float) \rightarrow None}

Set the window’s opacity.

The value must be between 0 and 1 inclusive.

\textbf{abstract cmd_place(x, y, width, height, borderwidth, bordercolor, above=False, margin=None) \rightarrow None}

Place the window with the given position and geometry.

\textbf{abstract cmd_resize_floating(dw: int, dh: int) \rightarrow None}

Add \(dw\) and \(dh\) to size of window

\textbf{abstract cmd_set_position(x: int, y: int) \rightarrow None}

Move floating window to \(x\) and \(y\); swap tiling window with the window under the pointer.

\textbf{abstract cmd_set_position_floating(x: int, y: int) \rightarrow None}

Move window to \(x\) and \(y\)
abstract cmd_set_size_floating(w: int, h: int) → None
    Set window dimensions to w and h

abstract cmd_static(screen: Optional[int] = None, x: Optional[int] = None, y: Optional[int] = None, width: Optional[int] = None, height: Optional[int] = None) → None
    Makes this window a static window, attached to a Screen.
    Values left unspecified are taken from the existing window state.

abstract cmd_toggle_floating() → None
    Toggle the floating state of the window.

abstract cmd_toggle_fullscreen() → None
    Toggle the fullscreen state of the window.

abstract cmd_toggle_maximize() → None
    Toggle the maximize state of the window.

abstract cmd_toggle_minimize() → None
    Toggle the minimize state of the window.

cmd_togroup(group_name: Optional[str] = None, groupName: Optional[str] = None, switch_group: bool = False) → None
    Move window to a specified group
    Also switch to that group if switch_group is True.
    groupName is deprecated and will be dropped soon. Please use group_name instead.

cmd_toscreen(index: Optional[int] = 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)

cmd_up_opacity() → None
    Increase the window's opacity by 10%.
3.4 Keybindings in images

3.4.1 Default configuration

![Qtile Keybindings Diagram](image-url)
3.4. Keybindings in images

```
3.4. Keybindings in images
```

```bash
Qtile Documentation, Release 0.20.1.dev0+g6bb5520.d20220124
```
3.4.2 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
4.1 Hacking on Qtile

4.1.1 Requirements

Here are Qtile's additional dependencies that may be required for tests:

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Ubuntu Package</th>
<th>Needed for</th>
</tr>
</thead>
<tbody>
<tr>
<td>pytest</td>
<td>python3-pytest</td>
<td>Running tests</td>
</tr>
<tr>
<td>PyGObject</td>
<td>python3-gi</td>
<td>Running tests (test windows)</td>
</tr>
<tr>
<td>Xephyr</td>
<td>xserver-xephyr</td>
<td>Testing with X11 backend (optional, see below)</td>
</tr>
<tr>
<td>mypy</td>
<td>python3-mypy</td>
<td>Testing qtile_check (optional)</td>
</tr>
<tr>
<td>imagemagick&gt;=6.8</td>
<td>imagemagick</td>
<td>test/test_images* (optional)</td>
</tr>
<tr>
<td>gtk-layer-shell</td>
<td>libgtk-layer-shell0</td>
<td>Testing notification windows in Wayland (optional)</td>
</tr>
<tr>
<td>dbus-launch</td>
<td>dbus-x11</td>
<td>Testing dbus-using widgets (optional)</td>
</tr>
<tr>
<td>notify-send</td>
<td>libnotify-bin</td>
<td>Testing Notify widget (optional)</td>
</tr>
<tr>
<td>xvfb</td>
<td>xvfb</td>
<td>Testing with X11 headless (optional)</td>
</tr>
</tbody>
</table>

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.

4.1.2 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
```
4.1.3 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.

4.1.4 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.

4.1.5 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: `pytest`
5. commit

Of course, your patches should also pass the unit tests as well (i.e. `make check`). These will be run by ci on every pull request so you can see whether or not your contribution passes.

4.1.6 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 will run a linter with our configuration over `libqtile` to ensure your patch complies with reasonable formatting constraints. We also request that git commit messages follow the `standard format`.

4.1.7 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.

### 4.1.8 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.

### 4.1.9 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.
4.1.10 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`.

4.1.11 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 virtualenv, 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).
4.1.12 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.

```json
{
    "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:

```bash
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.

4.1.13 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

4.1.14 Troubleshoot

Cairo errors

When running the Xephyr script (.scripts/xephyr), you might see tracebacks with attribute errors like the following or similar:

```
AttributeError: cffi library 'libcairo.so.2' has no function, constant or global_
--variable named 'cairo_xcb_surface_create'
```

If it happens, it might be because the cairocffi and xcffib dependencies were installed in the wrong order.

To fix this:
1. uninstall them from your environment: with pip uninstall cairocffi xcffib if using a virtualenv, or with your system package-manager if you installed the development version of Qtile system-wide.

2. re-install them sequentially (again, with pip or with your package-manager):

   ```bash
   pip install xcffib
   pip install --no-cache-dir cairocffi
   ```

See this issue comment for more information.

If you are using your system package-manager and the issue still happens, the packaging of cairocffi might be broken for your distribution. Try to contact the persons responsible for cairocffi's packaging on your distribution, or to install it from the sources with xcffib available.

### Fonts errors

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:

```bash
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.

## 4.2 Contributing

### 4.2.1 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. Described the issue, step-by-step, so that it is easy to reproduce and fix.
2. **Specific.** Do not write a essay about the problem. Be Specific and to the point. Try to summarize the problem in minimum words yet in effective way. Do not combine multiple problems even 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.

### 4.2.2 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](#).

---

**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 PEP8 and is formatted by [black](#).
- **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 `migrate.py` for examples and consult the [bowler documentation](#) for detailed help and documentation.
- **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 separate 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!

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).

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).

The following examples show how to run tests locally:

- To run the functional tests, use tox -e py39 (or a different environment). You can specify to only run a specific test file or even a specific test within that file with the following commands:

  tox -e py39 # Run all tests with python 3.9 as the interpreter  
  tox -e py39 -- -x test/widgets/test_widgetbox.py # run a single file  
  tox -e py39 -- -x test/widgets/test_widgetbox.py::test_widgetbox_widget

- 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!
5.1 Frequently Asked Questions

5.1.1 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...
```

5.1.2 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.
5.1.3 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)
```

5.1.4 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.

5.1.5 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 dependet on your system configuration where to do this. e.g. ArchLinux with libreoffice-fresh in `/etc/profile.d/libreoffice-fresh.sh`.

5.1.6 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].to_screen()`, use this:

```python
def go_to_group(name: str) -> Callable:
    def _inner(qtile: Qtile) -> None:
        if len(qtile.screens) == 1:
            qtile.groups_map[name].cmd_to_screen()
            return

        if name in '123':
            qtile.focus_screen(0)
            qtile.groups_map[name].cmd_to_screen()
        else:
            qtile.focus_screen(1)
            qtile.groups_map[name].cmd_to_screen()

        return _inner

for i in groups:
    keys.append(Key([mod], i.name, lazy.function(go_to_group(i.name))))
```
If you use the `GroupBox` widget you can make it reflect this behaviour:

```python
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:

```python
@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()
```

### 5.1.7 Where can I find example configurations and other scripts?

Please visit our [qtile-examples](https://github.com/qtile/qtile-examples) repo which contains examples of users’ configurations, scripts and other useful links.

### 5.2 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.
6.1 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.

6.1.1 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).

6.1.2 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.

```python
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.
_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.

_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.

_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).

---

_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.

6.1.3 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.
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:

```python
self.add_defaults(base.PaddingMixin.defaults)
```

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:

```python
self.add_defaults(base.MarginMixin.defaults)
```

6.1.4 Configuration

Now you know which class to base your widget on, you need to know how the widget gets configured.

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.

```python
defaults = [
    ("parameter_name",
     default_parameter_value,
     "Short text explaining what parameter does")
]
```

Users can override the default value when creating their config.py file.

```python
MyCustomWidget(parameter_name=updated_value)
```

Once the widget is initialised, these parameters are available at self.parameter_name.

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.
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__.

6.1.5 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.

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.

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
    None,      # Font shadow
    )
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.

```python
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)
```

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: 📢

**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.
6.1.6 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 are summarised below.

Timers

A non-blocking timer can be called by using the `self.timeout_add` method.

```python
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.

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:

```python
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.

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.

6.1.7 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)
        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:

```python
class MouseOverClock(widget.Clock):
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()
```

### 6.1.8 Debugging

You can use the `logger` object to record messages in the Qtile log file to help debug your development.

```python
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.
6.1.9 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)

6.1.10 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.

6.1.11 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.

6.2 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.
6.2.1 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`.

6.2.2 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.

6.2.3 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.

### 6.2.4 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:

```bash
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.

### 6.2.5 I was told to edit a commit message

If you need to edit the commit message of the last commit you did, use:

```bash
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 `rewind` command in the editor.

- genindex
cmd_static() (libqtile.backend.base.Window method), 131

cmd_status() (libqtile.core.manager.Qtile method), 126

cmd_switch_groups() (libqtile.core.manager.Qtile method), 126

cmd_switchgroup() (libqtile.core.manager.Qtile method), 126

cmd_sync() (libqtile.core.manager.Qtile method), 126

cmd_to_layout_index() (libqtile.core.manager.Qtile method), 126

cmd_to_screen() (libqtile.core.manager.Qtile method), 126

cmd_toggle_floating() (libqtile.backend.base.Window method), 131

cmd_toggle_fullscreen() (libqtile.backend.base.Window method), 131

cmd_toggle_group() (libqtile.config.Screen method), 129

cmd_toggle_maximize() (libqtile.backend.base.Window method), 131

cmd_toggle_minimize() (libqtile.backend.base.Window method), 131

cmd_togroup() (libqtile.backend.base.Window method), 131

delgroup() (libqtile.hook.subscribe method), 46

DF (class in libqtile.widget), 73

Dmenu (class in libqtile.extension), 42

DmenuRun (class in libqtile.extension), 42

Drag (class in libqtile.config), 19

DropDown (class in libqtile.config), 13

define() (libqtile.hook.subscribe method), 46

DNS (class in libqtile.widget), 73

E

enter_chord() (libqtile.hook.subscribe method), 46

EzConfig (class in libqtile.config), 18

F

float_change() (libqtile.hook.subscribe method), 46

Floating (class in libqtile.layout.floating), 48

focus_change() (libqtile.hook.subscribe method), 46

G

Gap (class in libqtile.bar), 23

GenPollText (class in libqtile.widget), 73

GenPollUrl (class in libqtile.widget), 74

GmailChecker (class in libqtile.widget), 74

Group (class in libqtile.config), 10

group_window_add() (libqtile.hook.subscribe method), 46

GroupBox (class in libqtile.widget), 75

H

HDDBusyGraph (class in libqtile.widget), 76

HDDGraph (class in libqtile.widget), 77

I

IdleRPG (class in libqtile.widget), 77

Image (class in libqtile.widget), 78

ImapWidget (class in libqtile.widget), 79

J

J4DmenuDesktop (class in libqtile.extension), 43

K

Key (class in libqtile.config), 17

KeyboardKbdd (class in libqtile.widget), 80

KeyboardLayout (class in libqtile.widget), 80

KeyChord (class in libqtile.config), 17

KhalCalendar (class in libqtile.widget), 81

Index
LaunchBar (class in libqtile.widget), 82
layout_change() (libqtile.hook.subscribe method), 46
leave_chord() (libqtile.hook.subscribe method), 47
Maildir (class in libqtile.widget), 82
Match (class in libqtile.config), 11
Matrix (class in libqtile.layout.matrix), 50
Max (class in libqtile.layout.max), 51
Memory (class in libqtile.widget), 83
MemoryGraph (class in libqtile.widget), 84
Mirror (class in libqtile.widget), 85
Moc (class in libqtile.widget), 85
MonadTall (class in libqtile.layout.xmonad), 51
MonadWide (class in libqtile.layout.xmonad), 53
Mpd2 (class in libqtile.widget), 86
Mpris2 (class in libqtile.widget), 89
Net (class in libqtile.widget), 89
net_wm_icon_change() (libqtile.hook.subscribe method), 47
NetGraph (class in libqtile.widget), 90
Notify (class in libqtile.widget), 91
NvidiaSensors (class in libqtile.widget), 91
OpenWeather (class in libqtile.widget), 92
Pomodoro (class in libqtile.widget), 95
Prompt (class in libqtile.widget), 95
PulseVolume (class in libqtile.widget), 96
Qtile (class in libqtile.core.manager), 122
QuickExit (class in libqtile.widget), 97
RatioTile (class in libqtile.layout.ratiotile), 55
restart() (libqtile.hook.subscribe method), 47
Rule (class in libqtile.config), 11
RunCommand (class in libqtile.extension), 44
ScratchPad (class in libqtile.config), 12
Screen (class in libqtile.config), 22
screen_change() (libqtile.hook.subscribe method), 47
screens_reconfigured() (libqtile.hook.subscribe method), 47
selection_change() (libqtile.hook.subscribe method), 47
selection_notify() (libqtile.hook.subscribe method), 47
Sep (class in libqtile.widget), 98
setgroup() (libqtile.hook.subscribe method), 47
She (class in libqtile.widget), 98
shutdown() (libqtile.hook.subscribe method), 47
simple_key_binder() (in module libqtile.dgroups), 11
Slice (class in libqtile.layout.slice), 56
Spacer (class in libqtile.widget), 99
Stack (class in libqtile.layout.stack), 56
startup() (libqtile.hook.subscribe method), 48
startup_complete() (libqtile.hook.subscribe method), 48
startup_once() (libqtile.hook.subscribe method), 48
StatusNotifier (class in libqtile.widget), 100
StockTicker (class in libqtile.widget), 105
SwapGraph (class in libqtile.widget), 101
Systray (class in libqtile.widget), 101
TaskList (class in libqtile.widget), 102
TextBox (class in libqtile.widget), 103
ThermalSensor (class in libqtile.widget), 103
ThermalZone (class in libqtile.widget), 104
Tile (class in libqtile.layout.tile), 56
TreeTab (class in libqtile.layout.tree), 57
VerticalTile (class in libqtile.layout.verticaltile), 59
Volume (class in libqtile.widget), 105
Wallpaper (class in libqtile.widget), 106
WidgetBox (class in libqtile.widget), 107
Window (class in libqtile.backend.base), 129
WindowCount (class in libqtile.widget), 108
WindowList (class in libqtile.extension), 44
WindowName (class in libqtile.widget), 108
WindowTabs (class in libqtile.widget), 109
Wlan (class in libqtile.widget), 110
Wttr (class in libqtile.widget), 110
Zoomy (class in libqtile.layout.zoomy), 60