qtile-extras

Release 0.24.1.dev0+g8ac6c78.d20240120

elParaguayo

Jan 20, 2024

GETTING STARTED

`qtile-extras` is a collection of mods made by elParaguayo for Qtile.

These are mods that I've made but which, for various reasons, may not end up in the main Qtile codebase.

They're designed for me but I've shared them in case they're of interest to anyone else.

Currently, this repo houses some *widgets* that I made as well as my "*popup toolkit*" which can be used to extend widgets or make standalone menus/launchers.

The new widgets are:

- *ALSAWidget*
- *AnalogueClock*
- *Bluetooth*
- *BrightnessControl*
- *ContinuousPoll*
- *CurrentLayoutIcon*
- *GithubNotifications*
- *GlobalMenu*
- *GroupBox2*
- *IWD*
- *Image*
- *LiveFootballScores*
- *Mpris2*
- *PulseVolume*
- *PulseVolumeExtra*
- *ScriptExit*
- *SnapCast*
- *StatusNotifier*
- *StravaWidget*
- *Syncthing*
- *TVHWidget*
- *UPowerWidget*
- *UnitStatus*
- *Visualiser*
- *Visualizer*
- *WiFiIcon*
- *WordClock*

There's a *mixin* if you want to add tooltips to widgets.

I've also added some "eye candy" in the form of:

- *Widget Decorations*

- *Wallpapers*

Lastly, I've created a new `ImgMask` class which, rather than drawing the source image, uses the source as a mask for the drawing. This can be used to change the colour of icons without needing to recrate the icons themselves. You can see the class *here*.

Note: These items are made primarily for my use and are not officially supported by Qtile. You are most welcome to install it and I hope that you find some parts of is useful. However, please note, I cannot guarantee that I will continue to maintain certain aspects of this repo if I am no longer using them so, be warned, things may break!

CHAPTER

ONE

INSTALLATION

Important: The git version of qtile-extras should only be installed alongside the git version of Qtile. This is because qtile-extras aims to main compatibility with the latest version.

If you are using the tagged release version of Qtile then you should use the matching tagged release of qtile-extras. These are guaranteed to be compatible but you will not be able to benefit from new features/bugfixes unless Qtile also publishes a new release.

1.1 PyPi

I have no current intentions to package this on PyPi. This means installation may be a bit more "manual" than for other packages.

1.2 Arch users

This is the easiest option as the package is in the AUR. Using your favourite helper, you just need to download and install the `qtile-extras` package (for the tagged release) or the `qtile-extras-git` package (for the latest git version).

1.3 Fedora

There is no official package for Fedora yet but you can install it from Copr:

```
dnf copr enable frostyx/qtile
dnf install qtile-extras
```

1.4 Everyone else

You can use `pip` to install the package e.g. `pip install --user .`.

Alternatively, you can use the `build` and `installer` modules and run:

```
python -m build --wheel
python -m installer dist/*.whl
```

CHAPTER

TWO

USING THE POPUP TOOLKIT

This guide explains how to create popups that can be used to add functionality to widgets or create standalone launchers.

2.1 What's in the toolkit?

The Toolkit has two types of object, a layout and a control. The layout is the container that helps organise the presentation of the popup. The controls are the objects that display the content.

A simple comparison would be to think of the `Bar` as the layout and widgets as the controls. However, a key difference of this toolkit is that the controls can be placed anywhere in a 2D space whereas widgets can only be ordered in one dimension.

2.2 Layouts

The toolkit provides three layouts: `PopupGridLayout`, `PopupRelativeLayout` and `PopupAbsoluteLayout`.

Descriptions and configuration options of these layouts can be found on *the reference page*.

2.3 Controls

Currently, the following controls are provided:

- `PopupText`: a simple text display object
- `PopupImage`: a control to display an image
- `PopupSlider`: a control to draw a line which marks a particular value (e.g. volume level)
- `PopupWidget`: a control to display a Qtile widget in the popup

Configuration options for these controls can be found on *the reference page*.

2.4 Callbacks

To add functionality to your popup, you need to bind callbacks to the individual controls. This is achieved in the same way as widgets i.e. a dictionary of `mouse_callbacks` is passed as a configuration option for the control. The control accepts any callable function but also accepts `lazy` objects like those used for key bindings.

2.5 Building a popup menu

Below is an example of creating a power menu in your `config.py`.

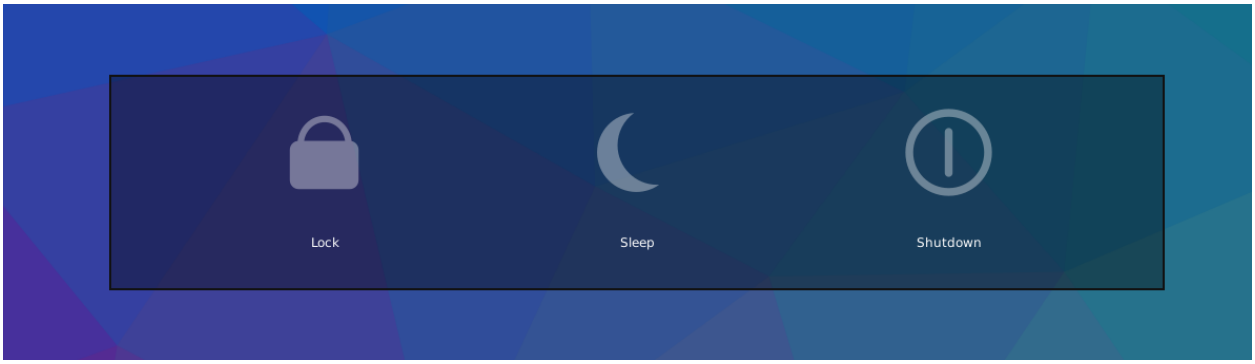
```
from qtile_extras.popup.toolkit import (
    PopupRelativeLayout,
    PopupImage,
    PopupText
)

def show_power_menu(qtile):

    controls = [
        PopupImage(
            filename="~/Pictures/icons/lock.svg",
            pos_x=0.15,
            pos_y=0.1,
            width=0.1,
            height=0.5,
            mouse_callbacks={
                "Button1": lazy.spawn("/path/to/lock_cmd")
            }
        ),
        PopupImage(
            filename="~/Pictures/icons/sleep.svg",
            pos_x=0.45,
            pos_y=0.1,
            width=0.1,
            height=0.5,
            mouse_callbacks={
                "Button1": lazy.spawn("/path/to/sleep_cmd")
            }
        ),
        PopupImage(
            filename="~/Pictures/icons/shutdown.svg",
            pos_x=0.75,
            pos_y=0.1,
            width=0.1,
            height=0.5,
            highlight="A00000",
            mouse_callbacks={
                "Button1": lazy.shutdown()
            }
        ),
        PopupText(
```

(continues on next page)

(continued from previous page)

```
            text="Lock",
            pos_x=0.1,
            pos_y=0.7,
            width=0.2,
            height=0.2,
            h_align="center"
        ),
        PopupText(
            text="Sleep",
            pos_x=0.4,
            pos_y=0.7,
            width=0.2,
            height=0.2,
            h_align="center"
        ),
        PopupText(
            text="Shutdown",
            pos_x=0.7,
            pos_y=0.7,
            width=0.2,
            height=0.2,
            h_align="center"
        ),
    ]

    layout = PopupRelativeLayout(
        qtile,
        width=1000,
        height=200,
        controls=controls,
        background="00000060",
        initial_focus=None,
    )

    layout.show(centered=True)

keys = [
    ...
    Key([mod, "shift"], "q", lazy.function(show_power_menu))
    ...
]
```

Now, when you press `Mod+shift+q` you should see a menu looking like this:



2.6 Using widgets in a popup

It is possible to display widgets in a popup window and not just in the bar. This is possible by using the `PopupWidget` control.

Below is a quick example for displaying a number of graph widgets in a popup:

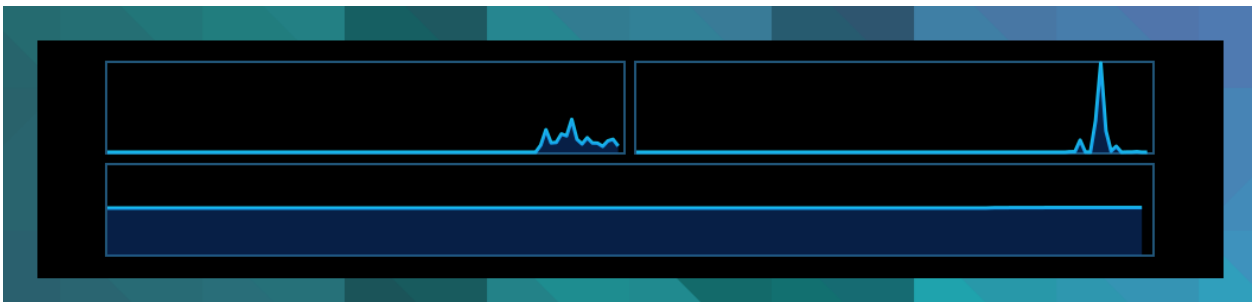
```
from libqtile import widget
from qtile_extras.popup.toolkit import (
    PopupRelativeLayout,
    PopupWidget
)

def show_graphs(qtile)
    controls = [
        PopupWidget(
            widget=widget.CPUGraph(),
            width=0.45,
            height=0.45,
            pos_x=0.05,
            pos_y=0.05
        ),
        PopupWidget(
            widget=widget.NetGraph(),
            width=0.45,
            height=0.45,
            pos_x=0.5,
            pos_y=0.05
        ),
        PopupWidget(
            widget=widget.MemoryGraph(),
            width=0.9,
            height=0.45,
            pos_x=0.05,
            pos_y=0.5
        )
    ]

    layout = PopupRelativeLayout(
        qtile,
```

(continues on next page)

(continued from previous page)

```
        width=1000,
        height=200,
        controls=controls,
        background="00000060",
        initial_focus=None,
        close_on_click=False
    )
    layout.show(centered=True)

keys = [
    ...
    Key([mod, "shift"], "g", lazy.function(show_graphs))
    ...
]
```

Pressing `Mod+shift+g` will present a popup window looking like this:



2.7 Updating controls

There may be times when you wish to update the contents of the popup without having to rebuild the whole popup. This is possible by using the `popup.update_controls` method. The method works by taking the name of the control (as set by the `name` parameter) as a keyword. Multiple controls can be updated in the same call.

```
from qtile_extras.popup.toolkit import (
    PopupRelativeLayout,
    PopupText
)

text_popup = None

def create_text_popup(qtile):
    global text_popup
    text_popup = PopupRelativeLayout(
        qtile,
        width=400,
        height=200,
        controls=[
            PopupText(
                text="Original Text",
                width=0.9,
```

(continues on next page)

(continued from previous page)

```
                height=0.9,
                pos_x=0.05,
                pos_y=0.05,
                v_align="middle",
                h_align="center",
                fontsize=20,
                name="textbox1"
            ),
        ],
        inital_focus=None,
        close_on_click=False,
    )

    text_popup.show(centered=True)

def update_text_popup(qtile):
    text_popup.update_controls(textbox1="Updated Text")
```

2.8 Extending widgets

To simplify the process of adding popup support to widgets, users are advised to use the `qtile_extras.widget.mixins.ExtendedPopupMixin` class as this will create a number of attributes and methods to help display popups.

For example, to make the `Clock` widget show the long date when clicked:

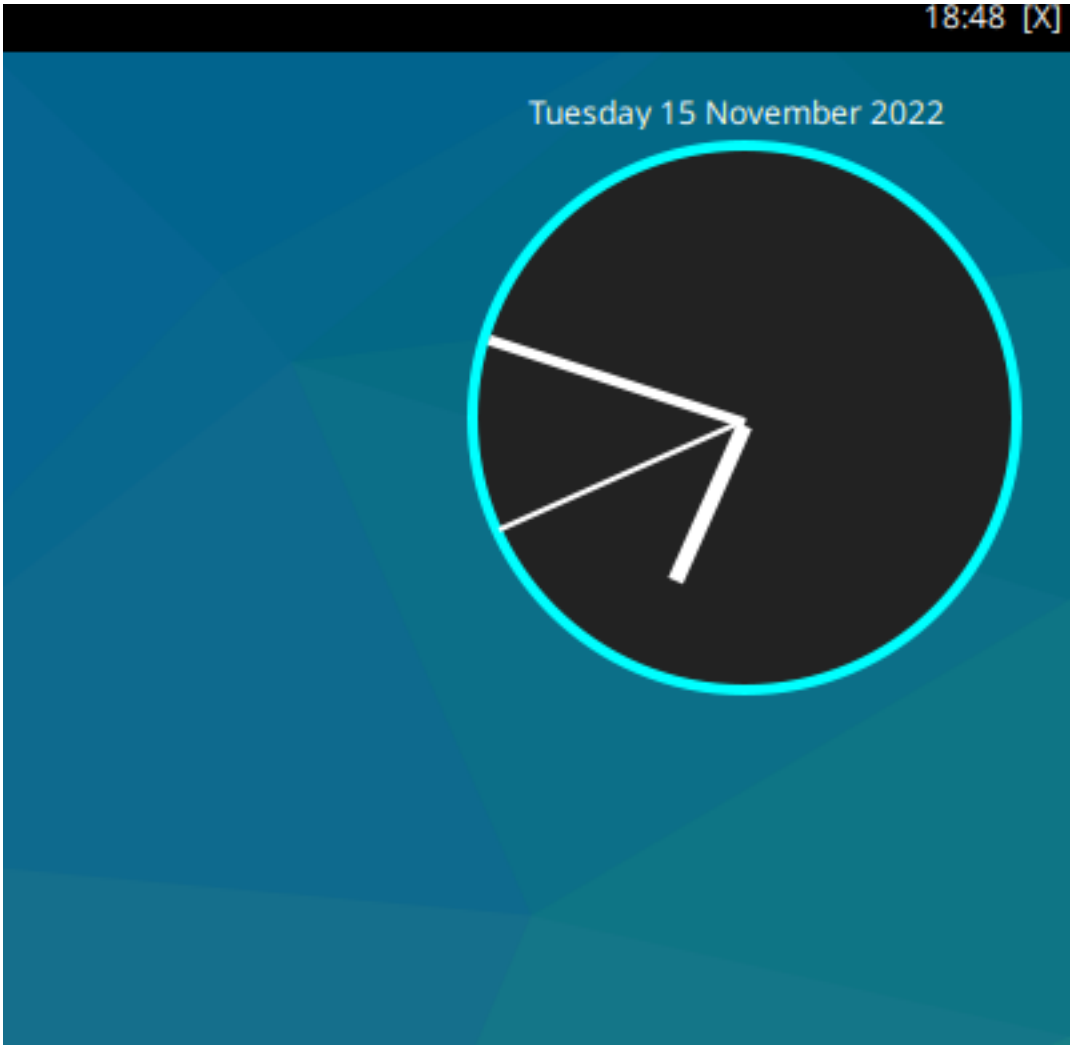
```
from datetime import datetime

from libqtile import widget

from qtile_extras import widget as extrawidgets
from qtile_extras.popup.toolkit import PopupRelativeLayout, PopupText, PopupWidget
from qtile_extras.widget.mixins import ExtendedPopupMixin


class ExtendedClock(widget.Clock, ExtendedPopupMixin):
    def __init__(self, **config):
        widget.Clock.__init__(self, **config)
        ExtendedPopupMixin.__init__(self, **config)
        self.add_defaults(ExtendedPopupMixin.defaults)
        self.add_callbacks({"Button1": self.show_popup})

    def _update_popup(self):
        longdate = datetime.now().strftime("%A %d %B %Y")
        self.extended_popup.update_controls(longdate=longdate)


clock_layout = PopupRelativeLayout(
    width=250,
    height=250,
    controls=[
        PopupText(
```

(continues on next page)

(continued from previous page)

```
            name="longdate",
            pos_x=0.1,
            pos_y=0.05,
            width=0.8,
            height=0.05,
            h_align="center",
        ),
        PopupWidget(
            widget=extrawidgets.AnalogueClock(
                second_size=2,
                minute_size=4,
                hour_size=6,
                face_shape="circle",
                face_background="222222",
                face_border_width=4,
            ),
            pos_x=0.05,
            pos_y=0.1,
            width=0.9,
            height=0.9,
        ),
    ],
    background="00000000",
)

extended_clock = ExtendedClock(
    popup_layout=clock_layout,
    popup_hide_timeout=0,
    popup_show_args={"relative_to": 3, "relative_to_bar": True},
)
```

Putting `extended_clock` in your bar and clicking on the clock will give you this:


18:48 [X]
Tuesday 15 November 2022

CHAPTER

THREE

WIDGET DECORATIONS

Widget decorations are additional content that is drawn to your widget before the main content is rendered i.e. you can add drawings behind your widgets.

3.1 Types of decoration

The following decorations are available:

- *BorderDecoration*
- *PowerLineDecoration*
- *RectDecoration*

3.2 Adding decorations to your widgets

All widgets available from this repo can have decorations added to them.

In addition, all widgets in the main Qtile repository can also have decorations attached. To do this, you simply need to change the import in your config file, replacing:

```
from libqtile import widget
```

with:

```
from qtile_extras import widget
```

A fuller example would look like this:

```
from qtile_extras import widget
from qtile_extras.widget.decorations import RectDecoration

decor = {
    "decorations": [
        RectDecoration(colour="#600060", radius=10, filled=True, padding_y=5)
    ],
    "padding": 18,
}

screens = [
```

(continues on next page)

(continued from previous page)

```
    Screen(
        bottom=bar.Bar(
            [
                widget.GroupBox(**decor),
                ...
                widget.Clock(**decor),
                widget.QuickExit(**decor),
            ]
        )
    )
]
```

3.3 Adding decorations to user-defined widgets

You can also add the ability to draw decorations to your own widgets.

First, you need to import `modify` from `qtile_extras.widget` and use this to wrap your widget class and its configuration parameters. i.e. calling `modify(WidgetClass, *args, **kwargs)` will return `WidgetClass(*args, **kwargs)`.

```
from libqtile.config import Screen
from libqtile.widget.base import _TextBox

from qtile_extras.bar import Bar
from qtile_extras.widget import modify


class MyTextWidget(_TextBox):
    pass


screens = [
    Screen(
        bottom=Bar(
            [
                ...
                modify(
                    MyTextWidget,
                    text="Modded widget",
                    decorations=[
                        ...
                    ]
                ),
                ...
            ]
        )
    )
]
```

CHAPTER

FOUR

IMAGE MASK

This is a new image class that allows you provide a source image to use as a mask. Painting with a colour then renders that colour in unmasked areas. The advantage of this is that the colour can be set dynamically without having to preload different images.

The example below shows a simple widget using this class to display three icons.

```
from libqtile import bar
from libqtile.widget.base import _Widget

from qtile_extras.images import ImgMask

ICON_PATH = "/path/to/icon_folder"

class MaskWidget(_Widget):
    def __init__(self):
        _Widget.__init__(self, bar.CALCULATED)

    def _configure(self, qtile, bar):
        _Widget._configure(self, qtile, bar)
        self.img = ImgMask.from_path(f"{ICON_PATH}/icon.svg")
        self.img.attach_drawer(self.drawer)
        self.img.resize(self.bar.height - 1)

    def calculate_length(self):
        if not self.configured:
            return 0

        return self.img.width * 3

    def draw(self):
        self.drawer.clear(self.background or self.bar.background)
        offset = 0
        for col in [
            "ff0000",
            "00ff00",
            ["ff00ff", "0000ff", "00ff00", "ff0000", "ffff00"]
        ]:
            self.img.draw(colour=col, x=offset)
            offset += self.img.width

        self.drawer.draw(
```

(continues on next page)

(continued from previous page)

```
            offsetx=self.offset,
            offsety=self.offsety,
            width=self.length
        )
```

Placing an instance of `MaskWidget()` in your bar will then give you something like this:

Fig. 1: Note you can use gradients too.

Note: It is important that the `ImgMask` object has a reference to the widget's `drawer` attribute. In the example above, this is achieved via the call to `self.img.attach_drawer(self.drawer)`.

4.1 Batch Loader

If you want to use the `Loader` class to load a batch of images to use as masks, you can do that as follows (note the use of the `masked=True` keyword argument):

```
from qtile_extras.images import Loader

image_dict = Loader(IMAGE_FOLDER, masked=True)(*IMAGE_NAMES)
```

As above, the images will need to have the widget's `drawer` object attached.

CHAPTER

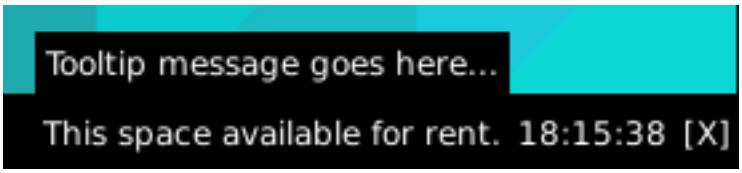
FIVE

WIDGET TOOLTIPS

Using the `TooltipMixin` allows you to add a tooltip to any widget. This is best illustrated with a simple example:

```
from libqtile.widget import TextBox

from qtile_extras.widget.mixins import TooltipMixin


class TooltipTextBox(TextBox, TooltipMixin):

    def __init__(self, *args, **kwargs):
        TextBox.__init__(self, *args, **kwargs)
        TooltipMixin.__init__(self, **kwargs)
        self.add_defaults(TooltipMixin.defaults)

        # The tooltip text is set in the following variable
        self.tooltip_text = "Tooltip message goes here..."

# Add an instance of TooltipTextBox to your bar
# e.g. TooltipTextBox("This space available for rent.")
```

When you hover your mouse over the widget you will see a message appear after a short delay:



See the *reference page* for instructions on how to customise the mixin.

CHAPTER

SIX

WIDGETS

6.1 ALSAWidget

class qtile_extras.widget.**ALSAWidget**(**args*, ***kwargs*)

The widget is very simple and, so far, just allows controls for volume up, down and mute.

Volume control is handled by running the appropriate amixer command. The widget is updated instantly when volume is changed via this code, but will also update on an interval (i.e. it will reflect changes to volume made by other programs).

The widget displays volume level via an icon, bar or both. The icon is permanently visible while the bar only displays when the volume is changed and will hide after a user-defined period.

Alternatively, if you select the *popup* mode then no widget will appear on the bar and, instead, a small popup will be displayed.

The layout of the popup can be customised via the *popup_layout* parameter. Users should provide a _PopupLayout object. The layout should have at least one of the following controls: a PopupSlider named *volume* and a PopupText control named *text* as these controls will be updated whenever the volume changes.

Supported bar orientations: horizontal only

Fig. 1: 'icon' mode

Fig. 2: 'bar' mode

Fig. 3: 'both' mode

key	default	description
background	None	Widget background color
bar_background	None	Colour of bar background.
bar_colour	'00ffff'	Colour of bar (NB this setting may be overridden by other widget settings).
bar_colour_high	'999900'	Colour of bar if high range
bar_colour_loud	'990000'	Colour of bar in loud range
bar_colour_mute	'999999'	Colour of bar if muted
bar_colour_norma	'009900'	Colour of bar in normal range

continues on next page

Table 1 – continued from previous page

key	default	description
bar_height	None	Height of bar (None = full bar height).
bar_text	''	Text to show over bar
bar_text_font	None	Font to use for bar text
bar_text_fontsiz	None	Fontsize for bar text
bar_text_foregro	'ffffff'	Colour for bar text
bar_width	75	Width of display bar
decorations	[]	Decorations for widgets
device	'Master'	Name of ALSA device
font	'sans'	Default font
fontsize	None	Font size
foreground	'ffffff'	Font colour
hide_interval	5	Timeout before bar is hidden after update
icon_size	None	Size of the volume icon
limit_high	90	Max percentage for high range
limit_loud	100	Max percentage for loud range
limit_normal	70	Max percentage for normal range
mode	'bar'	Display mode: 'icon', 'bar', 'both', 'popup'.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	0	Padding before icon
popup_hide_timeo	5	Time before popup hides
popup_layout	<qtile_extras.popup.toolkit.PopupRelativeLay object at 0x7f07c74f15d0>	Layout for popup mode
popup_show_args	{'relative_to': 2, 'relative_to_bar True, 'y': 50}	Control position of popup
step	5	Amount to increase volume by
text_format	'{volume}%'	String format
theme_path	None	Path to theme icons.
update_interval	5	Interval to update widget (e.g. if changes made in other apps).

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

show_popup()

Method to display the popup.

toggle_mute(**args*, ***kwargs*)

Mute audio output

volume_down(**args*, ***kwargs*)

Decrease volume

volume_up(**args*, ***kwargs*)

Increase volume

6.2 AnalogueClock

class qtile_extras.widget.**AnalogueClock**(**args*, ***kwargs*)

An analogue clock for your Bar.

The size of the clock will be the size of the bar minus 2x the margin. Use `padding` to add spacing before and after the widget. Finally, the position can be fine adjusted using the `adjust_x/y` values.

Supported bar orientations: horizontal and vertical

Fig. 4: Default config

Fig. 5: With square clock face

key	default	description
adjust_x	0	Adjust the x position of the widget
adjust_y	0	Adjust the y position of the widget
background	None	Widget background color
decorations	[]	Decorations for widgets
face_background	None	Shading for clock face
face_border_colo	'00ffff'	Border colour for clock face
face_border_widt	1	Thickness of clock face border
face_shape	None	'square', 'circle' or None
hour_colour	'ffffff'	Colour for the hour hand
hour_length	0.6	Length of hour hand as percentage of radius
hour_size	2	Thickness of hour hand
margin	2	Margin around clock
minute_colour	'ffffff'	Colour for the minute hand
minute_length	0.95	Length of minute hand as percentage of radius
minute_size	2	Thickness of minute hand
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	2	Additional padding at edges of widget
second_colour	'ffffff'	Colour for the second hand
second_length	0.95	Length of minute hand as percentage of radius
second_size	0	Thickness of second hand, 0 to hide.
update_interval	1	Polling interval in secs.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.3 Bluetooth

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

Modified version of the stock Qtile widget.

The only difference is to add a context menu (on Button 3) to show options for all adapters and devices.

Supported bar orientations: horizontal and vertical

key	default	description
adapter_format	'Adapter: {name} [{powered}{disco	Text to display when showing adapter device.
adapter_paths	[]	List of DBus object path for bluetooth adapter (e.g. '/org/bluez/hci0'). Empty list will show all adapters.
background	None	Widget background color
decorations	[]	Decorations for widgets
default_show_bat	False	Include battery level of 'connected_devices' in 'default_text'. Uses 'device_battery_format'.
default_text	'BT {connected_devic	Text to show when not scrolling through menu. Available fields: 'connected_devices' list of connected devices, 'num_connected_devices' number of connected devices, 'adapters' list of bluetooth adapters, 'num_adapters' number of bluetooth adapters.
default_timeout	None	Time before reverting to default_text. If 'None', text will stay on selected item.
device	None	Device path, can be found with d-feet or similar dbus explorer. When set, the widget will default to showing this device status.
device_battery_f	' ({battery}%)'	Text to be shown if device reports battery level
device_format	'Device: {name}{battery_l [{symbol}]'	Text to display when showing bluetooth device. The {adapter field is also available if you're using multiple adapters.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
hci	None	(deprecated) same as 'device'.
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
hide_unnamed_dev	False	Devices with no name will be hidden from scan results
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_theme	None	Icon theme for DBus menu items
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
markup	True	Whether or not to use pango markup

continues on next page

Table 2 – continued from previous page

key	default	description
max_chars	0	Maximum number of characters to display in widget.
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	200	Context menu width
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
opacity	1	Menu opactity
padding	None	Padding. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's `width`.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When `scroll=True` the `width` parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting `scroll_fixed_width=True` will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator	', '	Separator for lists in 'default_text'.
separator_colour	'555555'	Colour of menu separator
show_menu_icons	True	Show icons in context menu
symbol_connected	'*'	Symbol to indicate device is connected
symbol_discovery	('D', '')	Symbols when adapter is discovering and not discovering
symbol_paired	'-'	Symbol to indicate device is paired but unconnected
symbol_powered	('*', '-')	Symbols when adapter is powered and unpowered.
symbol_unknown	'?'	Symbol to indicate device is unpaired

click()

Perform default action on visible item.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

scroll_down()

Scroll down to next item.

scroll_up()

Scroll up to next item.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

show_devices()

Show menu with available adapters and devices.

6.4 BrightnessControl

class qtile_extras.widget.**BrightnessControl**(**args*, ***kwargs*)

This module provides basic screen brightness controls and a simple widget showing the brightness level for Qtile.

Brightness control is handled by writing to the appropriate `/sys/class/backlight` device. The widget is updated instantly when the brightness is changed via this code and will autohide after a user-defined timeout.

Note: This script will not work unless the user has write access to the relevant backlight device.

This can be achieved via a udev rule which modifies the group and write permissions. The rule should be saved at /etc/udev/rules.d

An example rule is as follows:

```
# Udev rule to change group and write permissions for screen backlight
ACTION=="add", SUBSYSTEM=="backlight", KERNEL=="intel_backlight", RUN+="/bin/chgrp␣
↪video /sys/class/backlight/%k/brightness"
ACTION=="add", SUBSYSTEM=="backlight", KERNEL=="intel_backlight", RUN+="/bin/chmod␣
↪g+w /sys/class/backlight/%k/brightness"
```

You should then ensure that your user is a member of the `video` group.

Supported bar orientations: horizontal only

key	default	description
background	None	Widget background color
bar_background	None	Colour of bar background.
bar_colour	'008888'	Colour of bar displaying brightness level.
bar_height	None	Height of bar (None = full bar height).
bar_text	''	Text to show over bar
bar_text_font	None	Font to use for bar text
bar_text_fontsiz	None	Fontsize for bar text
bar_text_foregro	'ffffff'	Colour for bar text
bar_width	75	Width of bar.
brightness_on_ba	'50%'	Brightness level on battery power (accepts integer value or percentage as string)
brightness_on_ma	'100%'	Brightness level on mains power (accepts integer valueor percentage as string)
brightness_path	'brightness'	Name of file holding brightness value
decorations	[]	Decorations for widgets
device	'/sys/class/backlight/intel_backlight'	Path to backlight device
enable_power_sav	False	Automatically set brightness depending on status. Note: this is not checked when the widget is first started.
error_colour	'880000'	Colour of bar when displaying an error
font	'sans'	Default font
fontsize	None	Font size
foreground	'ffffff'	Colour of text.
max_brightness	None	Set value or leave as None to allow device maximum
max_brightness_p	'max_brightness'	Name of file holding max brightness value
min_brightness	100	Minimum brightness. Do not set to 0!
mode	'bar'	Display mode: 'bar' shows bar in widget, 'popup' to display a popup window
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
popup_hide_timeo	5	Time before popup hides
popup_layout	<qtile_extras.popup.toolkit.PopupRelativeLayout object at 0x7f07c7570fd0>	Layout for popup mode

continues on next page

Table 3 – continued from previous page

key	default	description
popup_show_args	{'relative_to': 2, 'relative_to_bar True, 'y': 50}	Control position of popup
step	'5%'	Amount to change brightness (accepts int or percentage as string)
text_format	'{percentage}%'	Text to display.
timeout_interval	5	Time before widet is hidden.

brightness_down()

Decrease the brightness level

brightness_up()

Increase the brightness level

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_brightness_percent(*percent*)

Set brightness to percentage (0.0-1.0) of max value

set_brightness_value(*value*)

Set brightess to set value

show_popup()

Method to display the popup.

6.5 CPUGraph

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

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

6.6 ContinuousPoll

class qtile_extras.widget.**ContinuousPoll**(**args*, ***kwargs*)

A widget for displaying the continuous output from a process.

Every time a new line is output, the widget will update.

The widget takes an optional `parse_line` parameter which should be a callable object accepting a `line` object. The object should return a string to be displayed in the widget. NB the line received by the object is a raw bytestring so you may want to `decode()` it before manipulating it. The default behaviour (i.e. with no function) is to run `.decode().strip()` on the text to remove any trailing new line character.

Supported bar orientations: horizontal and vertical

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

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

run_process(*command=None*)

Re-run the command or provide a new command to run.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

stop_process(*kill=False*)

Stop the running process.

6.7 CurrentLayoutIcon

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

A modified version of Qtile's `CurrentLayoutIcon`.

The default version behaves the same as the main qtile version of the widget. However, if you set `use_mask` to `True` then you can set the colour of the icon via the `foreground` parameter.

Supported bar orientations: horizontal only

Fig. 6: You can use a single colour or a list of colours.

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

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

6.8 GithubNotifications

class qtile_extras.widget.**GithubNotifications**(**args*, ***kwargs*)

A widget to show when you have new github notifications.

The widget requires a personal access token (see here). The token needs the `notifications` scope to be enabled. This token should then be saved in a file and the path provided to the `token_file` parameter.

If your key expires, re-generate a new key, save it to the same file and then call the `reload_token` command (e.g. via `qtile cmd-obj`).

Required Dependencies

This module requires the following third-party libraries: `requests`

Supported bar orientations: horizontal only

Available hooks:

- ghn_new_notification

key	default	description
active_colour	'00ffff'	Colour when there are notifications
background	None	Widget background color
decorations	[]	Decorations for widgets
error_colour	'ffff00'	Colour when client has an error (check logs)
icon_size	None	Icon size. None = autofit.
inactive_colour	'ffffff'	Colour when there are no notifications.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	2	Padding around icon.
token_file	'~/.config/qtile-extras/github.token'	Path to file containing personal access token.
update_interval	150	Number of seconds before checking status.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

reload_token()

Force reload of access token.

6.9 GlobalMenu

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

`class qtile_extras.widget.GlobalMenu(*args, **kwargs)`

A widget to display a Global Menu (File Edit etc.) in your bar.

Only works with apps that export their menu via DBus.

Wayland support is "experimental" at best. Expect unexpected behaviour!

Required Dependencies

This module requires the following third-party libraries: `dbus-next`

Supported bar orientations: horizontal only

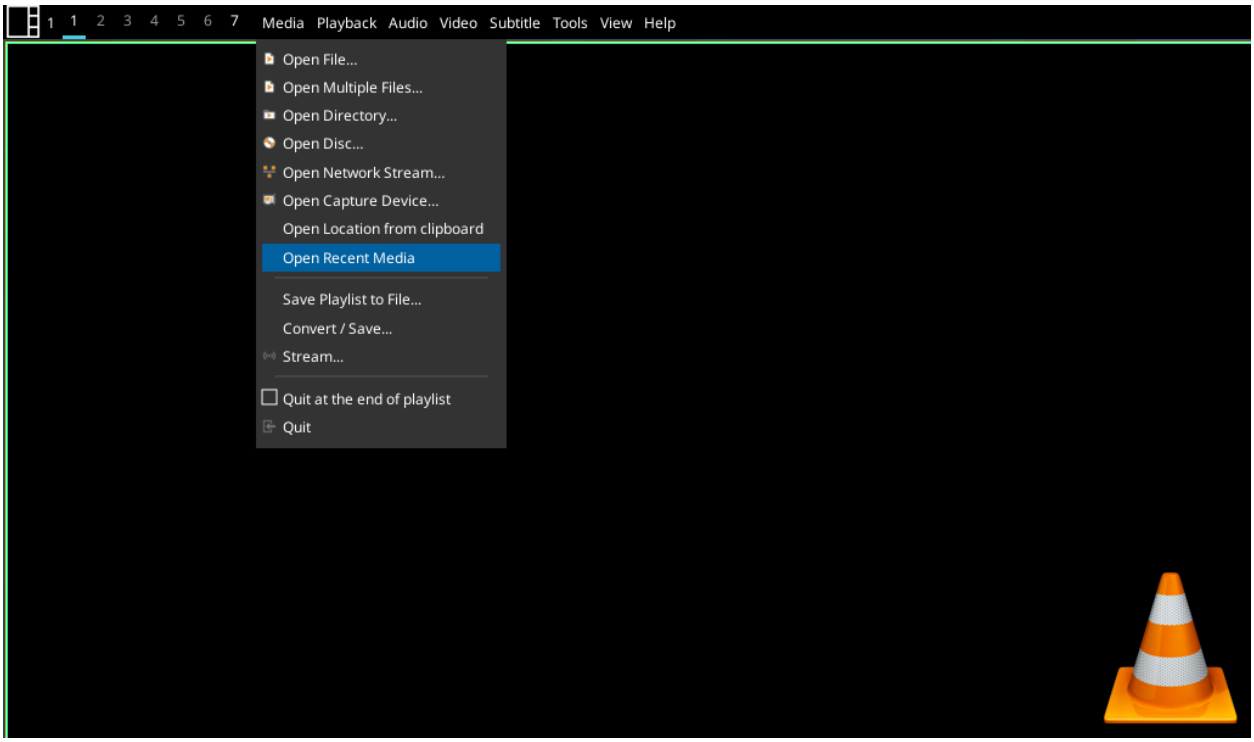


Fig. 7: Showing VLC menu in the bar.

key	default	description
`background`	`None`	Widget background color
`decorations`	`[]`	Decorations for widgets

continues on next page

Table 4 – continued from previous page

key	default	description
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_theme	None	Icon theme for DBus menu items
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	200	Context menu width
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
opacity	1	Menu opactity
padding	3	Padding between items in menu bar
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator_colour	'555555'	Colour of menu separator

continues on next page

Table 4 – continued from previous page

key	default	description
show_menu_icons	True	Show icons in context menu

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

6.10 GroupBox2

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

class qtile_extras.widget.**GroupBox2**(**args*, ***kwargs*)

Formatting of the group box is determined by applying user-defined rules to each group.

Overview:

A rule can set any combination of the following formats:

- text_colour - a string representing the hex value of the colour for the text

- block_colour - a string or list of strings to fill a block
- block_border_width - an integer representing the width of a block border
- block_border_colour - a string representing the colour of the block border
- block_corner_radius - an integer representing radius for curved corners
- line_colour - a string representing the colour of a line
- line_width - an integer representing the width of the line
- line_position - a flag representing where the line should be drawn
- image - path to an image file
- custom_draw - a function that draws to the box
- text - string representing text to display
- box_size - integer to force the size of the individual box

Whether a rule is applied will depend on whether it meets the relevant conditions set for each rule. A rule can set any combination of the following conditions:

- Which screen the group is on (same screen as bar, different screen, no screen)
- Whether the group is focused (i.e. the current group) - boolean True/False
- Whether the group has windows - boolean True/False
- Whether the group has any urgent windows - boolean True/False
- Whether the group name matches a given string
- Whether a user-defined function returns True

Order of drawing:

The widget draws the groupbox in the following order:

- Background colour
- Block
- Block border
- Line
- Image
- Custom draw function
- Text

Explanation of groupbox items:

Block:

Block is a rectangle that can be filled (with *block_colour*) and/or have an outline (with *block_border_width* and *block_border_colour*). The corners of the rectangle can be curved by setting the *block_corner_radius* value.

The block is positioned by using the *margin(_x)* and *margin_y* attributes. NB Currently, these are global for the widget and cannot be set by rules.

Line:

Line is a straight line on the edge of the widget. A line will be drawn at the bottom of the box by default (when a *line_colour* and *line_width have been set). The position of lines can be changed by setting the `line_position*

attribute with a LinePosition flag. For example to drawn lines at the top and bottom of the box you would set the *line_position* value to:

```
GroupBoxRule.LINE_TOP | GroupBoxRule.LINE_BOTTOM
```

NB the line will be rendered at the edge of the box i.e. it is not currently offset by the margin value.

Image:

Image renders any image file in the box. The image will be shrunk so that the height fits within the bar and is also adjusted by the `margin(_x)` and `margin(_y)` attributes.

Text:

A rule is able to set custom text for a group box. Where this is not set, the box will display the group's label or name by default. Setting a value of `None` will prevent text from being shown.

Custom draw:

You can define a function to draw directly to the box. The function should take a single argument which is the instance of the box. You can access the drawer object and its context via the `box.drawer` and `box.drawer.ctx` attributes.

The drawing should be constrained to the rectangle defined by (x=0, y=0, width=box.size, height=box.bar.height). The origin is the top left corner.

For example, to define a rule that draws a red square in the middle of the box for occupied groups, you would do the following:

```
def draw_red_square(box):

    w = 10
    h = 10
    x = (box.size - w) // 2
    y = (box.bar.height - h) // 2
    box.drawer.ctx.rectangle(x, y, w, h)
    box.drawer.set_source_rgb("ff0000")
    box.drawer.ctx.fill()

# Add this to your rules:
GroupBoxRule(custom_draw=draw_red_square).when(occupied=True)
```

Creating rules:

Creating a rule has two steps:

- Setting the desired format
- Setting the conditions required for that rule

To help readibility of config files, this is split so the format is set in the rule's constructor while the conditions are set by the rule's `when()` method.

For example, to set a rule that sets the font colour to cyan when a group has windows but is not focused, you would create the following rule:

```
GroupBoxRule(text_colour="00ffff").when(focused=False, occupied=True)
#                    ^                              ^
#      Format set via constructor    Conditions set via when() method
```

How to match conditions:

Screen:

Matching a screen condition uses a ScreenRule object. Available options are:

- GroupBoxRule.SCREEN_UNSET (default) - rule ignores screen condition
- GroupBoxRule.SCREEN_THIS - group is on same screen as widget's bar
- GroupBoxRule.SCREEN_OTHER - group is a different screen to the widget's bar
- GroupBoxRule.SCREEN_ANY (same as GroupBoxRule.SCREEN_THIS | GroupBoxRule.SCREEN_OTHER) - group is on any screen
- GroupBoxRule.SCREEN_NONE - group is not displayed on a screen

Group focus:

You can match a rule if the group is focused or unfocused by setting `focused` to `True` or `False` respectively. Leaving this attribute blank will ignore focus.

Group has windows:

You can match a rule if the group has windows or is empty by setting `occupied` to `True` or `False` respectively. Leaving this attribute blank will ignore the contents of a group.

Group has urgent windows:

You can match a rule if the group has or does not have any ugent windows by setting `urgent` to `True` or `False` respectively. Leaving this attribute blank will ignore the urgency state of a group.

Match group name:

You can tie a rule to a specific group by setting `group_name` to the name of a particular group. Leaving this blank will ignore the group's name.

User-defined functions:

Using custom functions can extend the possibilities for customising the widget. These are set via the `func` argument.

A function must take two arguments (the *GroupBoxRule* object and the *Box* object (that draws the specific box)) and return a boolean (True if the rule should be applied or False if not).

By accessing properties of the *Box* object, it is possible to fine tune the criteria against which the rule will match. The *Box* has the following attributes which may be of use here: *qtile* - the qtile object, *group* - the specific group represented by the box, *bar* - the Bar containing this widget.

As an example, to create a rule that is matched when a specific app is open in the group:

```
def has_vlc(rule, box):
    for win in box.group.windows:
        if "VLC" in win.name:
            return True

    return False

# Include this in your group box rules
# Turns label a nice shade of orange if the group has a VLC window open
GroupBoxRule(text_colour="E85E00").when(func=has_vlc)
```

In addition, a user-defined function can set display properties dynamically. For example, to have a different icon depending on the state of the group:

```
def set_label(rule, box):
    if box.focused:
        rule.text = ""
    elif box.occupied:
        rule.text = ""
    else:
        rule.text = ""

    return True

# Include this in your group box rules
# NB: The function returns True so this rule will always be run
GroupBoxRule().when(func=set_label)
```

Rule hierarchy:

Rules are applied in a hierarchy according to the order that they appear in the *rules* parameter. Once a format has been set by a rule, it cannot be updated by a later rule. However, if one rule sets the text colour a later rule can still set another format (e.g. border colour). To prevent lower priority rules from setting a value, a rule can set the *None* value for that attribute.

For example:

```
rules = [
    GroupBoxRule(block_colour="009999").when(screen=GroupBoxRule.SCREEN_THIS),
    GroupBoxRule(block_colour="999999").when(occupied=True),
    GroupBoxRule(text_colour="ffffff")
]
```

This rule will set all text white but will apply a blue block when the group is shown on the same screen as the widget even if the group is occupied. An occupied group will have a grey background provided it's not on the same screen as the widget.

Supported bar orientations: horizontal only

key	default	description
background	None	Widget background color
current_screen_f	None	Sets a fixed width for the focused group on the current screen.
decorations	[]	Decorations for widgets
font	'sans'	Font to use for label.
fontshadow	None	Font shadow
fontsize	None	Fontsize for labels,
invert_mouse_whe	False	Whether to invert mouse wheel group movement
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
markup	False	Use markup.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	3	Padding inside the box
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
rules	[<GroupBoxRule format(line_colo when(screen=Scre THIS)>, <GroupBoxRule format(line_colo when(screen=Scre OTHER)>, <GroupBoxRule format(text_colo when(occupied=Tr <GroupBoxRule format(text_colo when(occupied=Fa	Rules for determining how to format group box. See docstring for fuller explanation.
use_mouse_wheel	True	Whether to use mouse wheel events
visible_groups	None	List of names of groups to show in widget. 'None' (default) to show all groups.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.11 HDDBusyGraph

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

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

6.12 HDDGraph

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

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

6.13 IWD

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

class qtile_extras.widget.**IWD**(**args*, ***kwargs*)

This widget provides information about your wireless connection using iwd.

The widget also allows you to scan for and connect to different networks. If the network is unknown (i.e. you haven't connected to it before), the widget will launch a window to enter the password (using zenity by default).

NB you cannot join 802.1x networks unless they have already been configured.

Supported bar orientations: horizontal and vertical

key	default	description
active_colour	'ffffff'	Colour for wifi strength.

continues on next page

Table 5 – continued from previous page

key	default	description
background	None	Widget background color
check_connection	0	Interval to check if device connected to internet (0 to disable)
decorations	[]	Decorations for widgets
disconnected_col	'aa0000'	Colour when device has no internet connection
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns `foo`, using `fmt='<i>{}</i>'` would give you `<i>foo</i>`. To control what the widget outputs in the first place, use the `format` paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{ssid} ({quality}%)'	Text format. Available fields: ssid, rssi, quality
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_theme	None	Icon theme for DBus menu items
inactive_colour	'666666'	Colour for wifi background.
interface	None	Name of wireless interface. You should only need to set this if you have more than one wireless adapter on your system.
internet_check_h	'8.8.8.8'	IP adddress to check for internet connection
internet_check_p	53	Port to check for internet connection
internet_check_t	5	Period before internet check times out and widget reports no internet connection.
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	200	Context menu width
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
opacity	1	Menu opactity

continues on next page

Table 5 – continued from previous page

key	default	description
padding	None	Padding. Calculated if None.
password_entry_a	'zenity'	Application for password entry.
password_entry_a	['--entry', '--text', 'Enter password:', '--hide-text']	Additional args to pass to password entry command.
scanning_colour	'3abb3a'	Colour to use for image when scanning is active.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator_colour	'555555'	Colour of menu separator
show_image	False	Shows a graphical representation of signal strength.
show_menu_icons	True	Show icons in context menu
show_text	True	Displays text in bar.
update_interval	2	Polling interval in seconds.
wifi_arc	75	Angle of arc in degrees.
wifi_rectangle_w	5	Width of rectangle in pixels.
wifi_shape	'arc'	'arc' or 'rectangle'

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

scan()

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

6.14 Image

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

A modified version of Qtile's `Image` widget.

The two key differences are: 1) The widget accepts a url and will download the image from the internet. 2) The image can be used as a mask and filled with a user-defined colour.

Supported bar orientations: horizontal and vertical

key	default	description
`adjust_x`	`0`	Fine x-axis adjustment of icon position
`adjust_y`	`0`	Fine y-axis adjustment of icon position
`background`	`None`	Widget background color
`colour`	`'ffffff'`	Colour to paint maksed image
`decorations`	`[]`	Decorations for widgets
`filename`	`None`	Image filename. Can contain '~'. Can also be a url.
`margin`	`3`	Margin inside the box
`margin_x`	`None`	X Margin. Overrides 'margin' if set
`margin_y`	`None`	Y Margin. Overrides 'margin' if set
`mask`	`False`	Use the image as a mask and fill with `colour`.
`mouse_callbacks`	`{}`	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
`padding`	`0`	Padding to left and right of image on horizontal bar, or above and below widget on vertical bar.
`rotate`	`0.0`	rotate the image in degrees counter-clockwise
`scale`	`True`	Enable/Disable image scaling

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

update(*filename*)

6.15 LiveFootballScores

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

class qtile_extras.widget.**LiveFootballScores**(**args*, ***kwargs*)

The module uses a module I wrote a number of years ago that parses data from the BBC Sport website.

The underlying module needs work so it will probably only work if you pick a "big" team.

You can select more than one team and league. Scores can be scrolled by using the mousewheel over the widget.

Goals and red cards are indicated by a coloured bar next to the relevant team name. The match status is indicated by a coloured bar underneath the match summary. All colours are customisable.

Right-clicking the widget will bring up a list of all matches that meet your selected criteria. Clicking on any of those matches will open a popup showing more detail.

The popup can be accessed directly by the `show_detail()` command. When this is used, the selected match is the one currently visible in the widget.

Required Dependencies

This module requires the following third-party libraries: `requests`

Supported bar orientations: horizontal only

Available hooks:

Fig. 8: The different screens show: live score, elapsed time, home and away goalscorers and competition name. In addition, the amount of text shown can be customised by using python's string formatting techniques e.g. the default line `{H:.3} {h}-{a} {A:.3}` shows the first 3 letters of team names rather than the full name as shown above.

- lfs_goal_scored
- lfs_status_change
- lfs_red_card

key	default	description
always_show_red	True	Continue to show red card indicator
background	None	Widget background color
decorations	[]	Decorations for widgets
font	'sans'	Default font
fontsize	None	Font size
foreground	'ffffff'	Text colour
goal_indicator	'009999'	Colour of line to show team that scores
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_theme	None	Icon theme for DBus menu items
info_text	['{T:^12}', '{H:.3}: {G:10}', '{A:.3}: {g:10}', '{C}']	n Add extra text lines which can be displayed by clicking on widget.n Available fields are:n {H}: Home Team namen {A}: Away Team namen {h}: Home scoren {a}: Away scoren {C}: Competitionn {v}: Venuen {T}: Display time (kick-off, elapsed time, HT, FT)n {S}: Status (as above but no elapsed time)n {G}: Home goalscorersn {g}: Away goalscorersn {R}: Home red cardsn {r}: Away red cardsn
info_timeout	5	Time before reverting to default text
leagues	[]	List of leagues you want to display
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	300	Width of menu showing all matches
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.

continues on next page

Table 6 – continued from previous page

key	default	description
opacity	0.8	Opacity for popup window.
popup_display_ti	10	Seconds to show recordings.
popup_font	'monospace'	Font to use for displaying upcoming recordings. A monospace font is recommended
popup_hide_timeo	0	Number of seconds before popup is hidden (0 to disable).
popup_layout	<qtile_extras.popup.toolkit.PopupRelativeLay object at 0x7f07c7444b80>	Layout to use for extended match information
popup_show_args	{'centered': True, 'hide_on_timeout 5}	Arguments to set behaviour of extended popup
popup_text	'{H:.20} {h}-{a} {A:.20} ({T:.5})'	Format to use for popup window.
red_card_indicat	'bb0000'	Colour of line to show team has had a player sent off.
refresh_interval	60	Time to update data
separator_colour	'555555'	Colour of menu separator
show_menu_icons	True	Show icons in context menu
startup_delay	30	Time before sending first web request
status_fixture	'000000'	Colour when match has not started
status_fulltime	'666666'	Colour when match has ended
status_halftime	'aaaa00'	Colour when half time
status_live	'008800'	Colour when match is live
status_text	'{H:.3} {h}-{a} {A:.3}'	Default widget match text
team	'Liverpool'	Team whose scores you want to display
teams	[]	List of other teams you want to display
underline_status	True	Bar at bottom of widget to indicate status.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

get()

Get displayed text. Removes padding.

info()

Show information about all matches

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

popup()

Display window listing all matches

reboot()

Sometimes the widget won't update (and I don't know why). This method should reset everything and start the widget again.

Can be bound to a key e.g.:
lazy.widget["livefootballscores"].reboot

refresh()

Force a poll of match data.

show_detail()

Displays popup showing detailed info about match.

show_popup()

Method to display the popup.

6.16 MemoryGraph

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

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

6.17 Mpris2

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

Modified version of the base Mpris2 widget.

This version adds a popup with player controls. Users can provide a custom template for the popup using the `popup_layout` parameter.

The popup can be toggled with the `toggle_player` command.

The following fields are available (controls should set their 'name' to this value):

- 'title': Track title
- 'artist': Track artist
- 'album': Album name
- 'player': Media player name
- 'artwork': Path to artwork (to be used with a PopupImage control)
- 'progress': Progress through thrack (to be used with a PopupSlider control)
- 'position': Current playback position e.g. '03:40'
- 'length': Track length e.g. '05:15'
- 'time': String showing position and total length e.g. '03:40 / 05:15'

To control playback, the template should have controls named:

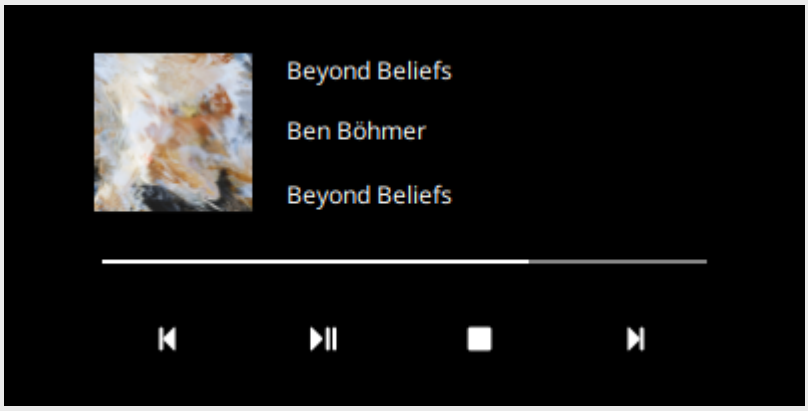
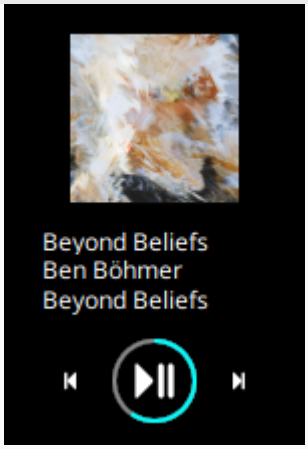
- 'play_pause'
- 'stop'
- 'previous'
- 'next'

When shown, the controls can be selected using the mouse or keyboard navigation. The popup can be hidden by pressing <escape> or by calling the `toggle_player` command.

Two pre-defined layouts are currently provided and can be loadeded via:

```
from qtile_extras import widget
from qtile_extras.popup.templates.mpris2 import COMPACT_LAYOUT, DEFAULT_LAYOUT

...

# NB DEFAULT_LAYOUT is included by default and does not need to be imported in
# your config
widget.Mpris2(popup_layout=COMPACT_LAYOUT)
```

The layouts look like this:

DEFAULT_LAYOUT	
COMPACT_LAYOUT	

Supported bar orientations: horizontal and vertical

Available hooks:

- mpris_new_track
- mpris_status_change

key	default	description
`background`	`None`	Widget background color
`decorations`	`[]`	Decorations for widgets
`default_artwork`	`'/home/docs/checkouts/readthedocs.org/user_builds/qtile-extras/checkouts/v0.24.0/qtile_extras/resources/media-icons/default.png'`	Image to display in popup when there's no art

continues on next page

Table 7 – continued from previous page

key	default	description
display_metadata	['xesam:title', 'xesam:album', 'xesam:artist']	(Deprecated) Which metadata identifiers to display.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
format	'{xesam:title} – {xesam:album} – {xesam:artist}'	Format string for displaying metadata. See http://www.freedesktop.org/wiki/Specifications/mpris-spec/metadata/#index5h3 for available values
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
name	'audacious'	Name of the MPRIS widget.
no_metadata_text	'No metadata for current track'	Text to show when track has no metadata
objname	None	DBUS MPRIS 2 compatible player identifier- Find it out with dbus-monitor - Also see: http://specifications.freedesktop.org/mpris-spec/latest/#Bus-Name-Policy. None will listen for notifications from all MPRIS2 compatible players.
padding	None	Padding. Calculated if None.
parse_artwork	<function parse_artwork at 0x7f07c7295a20>	Function to parse artwork path.
paused_text	'Paused: {track}'	Text to show when paused
playing_text	'{track}'	Text to show when playing
poll_interval	0	Periodic background polling interval of player (0 to disable polling).
popup_hide_timeo	0	Number of seconds before popup is hidden (0 to disable).
popup_layout	<qtile_extras.popup.toolkit.PopupRelativeLay object at 0x7f07c72a4730>	Layout for player controls.
popup_show_args	{'relative_to': 2, 'relative_to_bar True}	Where to place popup
scroll	True	Whether text should scroll.

continues on next page

Table 7 – continued from previous page

key	default	description
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator	', '	Separator for metadata fields that are a list.
stop_pause_text	None	(Deprecated) Optional text to display when in the stopped/paused state
stopped_text	''	Text to show when stopped

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

What's the current state of the widget?

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

next() → None

Play the next track.

play_pause() → None

Toggle the playback status.

previous() → None

Play the previous track.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

show_popup()

Method to display the popup.

stop() → None

Stop playback.

toggle_player()

6.18 NetGraph

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

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

6.19 PulseVolume

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

Same as qtile's `PulseVolume` widget but includes the ability to select the default output sink via the `select_sink()` command. This is bound to the middle-click button on the widget by default.

Supported bar orientations: horizontal only

key	default	description
background	None	Widget background color
cardid	None	Card Id
channel	'Master'	Channel
check_mute_comma	None	Command to check mute status
check_mute_strin	'[off]'	String expected from check_mute_command when volume is muted.When the output of the command matches this string, theaudio source is treated as muted.
decorations	[]	Decorations for widgets
device	'default'	Device Name
emoji	False	Use emoji to display volume states, only if `theme_path` is not set.The specified font needs to contain the correct unicode characters.
emoji_list	['', '', '', '']	List of emojis/font-symbols to display volume states, only if `emoji` is set. List contains 4 symbols, from lowest volume to highest.

continues on next page

Table 8 – continued from previous page

key	default	description
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size. Calculated if None.
foreground	'ffffff'	Foreground colour
get_volume_comma	None	Command to get the current volume. The expected output should include 1-3 numbers and a % sign.
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_theme	None	Icon theme for DBus menu items
limit_max_volume	False	Limit maximum volume to 100%
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	300	Width of list showing available sinks.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
mute_command	None	Mute command
opacity	1	Menu opactity
padding	3	Padding left and right. Calculated if None.
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished

continues on next page

Table 8 – continued from previous page

key	default	description
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator_colour	'555555'	Colour of menu separator
show_menu_icons	True	Show icons in context menu
step	2	Volume change for up an down commands in percentage.Only used if volume_up_command and volume_down_command are not set.
theme_path	None	Path of the icons
update_interval	0.2	Update time in seconds.
volume_app	None	App to control volume
volume_down_comm	None	Volume down command
volume_up_comman	None	Volume up command

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_vol(*value=None*)

Decrease volume.

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

increase_vol(*value=None*)

Increase volume.

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

mute()

Mute the sound device.

run_app()

select_sink()

Select output sink from available sinks.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

6.20 PulseVolumeExtra

class qtile_extras.widget.**PulseVolumeExtra**(**args*, ***kwargs*)

Volume widget for systems using PulseAudio.

The appearance is identical to `ALSAWidget` but this widget uses qtile's `PulseVolume` widget to set/retrieve volume levels. As a result, you will need the pulsectl_asyncio library to use this widget.

Finally, the widget allows users to select the output sink by middle clicking on the widget or calling the `select_sink()` command.

Supported bar orientations: horizontal only

Fig. 9: 'icon' mode

Fig. 10: 'bar' mode

Fig. 11: 'both' mode

key	default	description
background	None	Widget background color
bar_background	None	Colour of bar background.
bar_colour	'00ffff'	Colour of bar (NB this setting may be overridden by other widget settings).
bar_colour_high	'999900'	Colour of bar if high range
bar_colour_loud	'990000'	Colour of bar in loud range
bar_colour_mute	'999999'	Colour of bar if muted
bar_colour_norma	'009900'	Colour of bar in normal range
bar_height	None	Height of bar (None = full bar height).
bar_text	''	Text to show over bar
bar_text_font	None	Font to use for bar text
bar_text_fontsiz	None	Fontsize for bar text
bar_text_foregro	'ffffff'	Colour for bar text
bar_width	75	Width of display bar
cardid	None	Card Id
channel	'Master'	Channel
check_mute_comma	None	Command to check mute status
check_mute_strin	'[off]'	String expected from check_mute_command when volume is muted.When the output of the command matches this string, theaudio source is treated as muted.

continues on next page

Table 9 – continued from previous page

key	default	description
decorations	[]	Decorations for widgets
device	'Master'	Name of ALSA device
emoji	False	Use emoji to display volume states, only if theme_path is not set.The specified font needs to contain the correct unicode characters.
emoji_list	['', '', '', '']	List of emojis/font-symbols to display volume states, only if emoji is set. List contains 4 symbols, from lowest volume to highest.
fmt	'{}'	Format to apply to the string returned by the widget. Main purpose: applying markup. For a widget that returns foo, using fmt='{}' would give you foo. To control what the widget outputs in the first place, use the format paramater of the widget (if it has one).
font	'sans'	Default font
fontshadow	None	font shadow color, default is None(no shadow)
fontsize	None	Font size
foreground	'ffffff'	Font colour
get_volume_comma	None	Command to get the current volume. The expected output should include 1-3 numbers and a % sign.
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
hide_interval	5	Timeout before bar is hidden after update
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_size	None	Size of the volume icon
icon_theme	None	Icon theme for DBus menu items
limit_high	90	Max percentage for high range
limit_loud	100	Max percentage for loud range
limit_max_volume	False	Limit maximum volume to 100%
limit_normal	70	Max percentage for normal range
markup	True	Whether or not to use pango markup
max_chars	0	Maximum number of characters to display in widget.
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) “None” will attempt to calculate height based on font size.
menu_width	300	Width of list showing available sinks.
mode	'bar'	Display mode: ‘icon’, ‘bar’, ‘both’, ‘popup’.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.

continues on next page

Table 9 – continued from previous page

key	default	description
mute_command	None	Mute command
opacity	1	Menu opactity
padding	0	Padding before icon
popup_hide_timeo	5	Time before popup hides
popup_layout	<qtile_extras.popup.toolkit.PopupRelativeLay object at 0x7f07c74f15d0>	Layout for popup mode
popup_show_args	{'relative_to': 2, 'relative_to_bar True, 'y': 50}	Control position of popup
scroll	False	Whether text should be scrolled. When True, you must set the widget's width.
scroll_clear	False	Whether text should scroll completely away (True) or stop when the end of the text is shown (False)
scroll_delay	2	Number of seconds to pause before starting scrolling and restarting/clearing text at end
scroll_fixed_wid	False	When scroll=True the width parameter is a maximum width and, when text is shorter than this, the widget will resize. Setting scroll_fixed_width=True will force the widget to have a fixed width, regardless of the size of the text.
scroll_hide	False	Whether the widget should hide when scrolling has finished
scroll_interval	0.1	Time in seconds before next scrolling step
scroll_repeat	True	Whether text should restart scrolling once the text has ended
scroll_step	1	Number of pixels to scroll with each step
separator_colour	'555555'	Colour of menu separator
show_menu_icons	True	Show icons in context menu
step	5	Amount to increase volume by
text_format	'{volume}%'	String format
theme_path	None	Path to theme icons.
update_interval	5	Interval to update widget (e.g. if changes made in other apps).
volume_app	None	App to control volume
volume_down_comm	None	Volume down command
volume_up_comman	None	Volume up command

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

decrease_vol(*value=None*)

Decrease volume.

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

increase_vol(*value=None*)

Increase volume.

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

mute()

Mute the sound device.

run_app()

select_sink()

Select output sink from available sinks.

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

show_popup()

Method to display the popup.

toggle_mute(**args*, ***kwargs*)

Mute audio output

volume_down(*value=None*)

Decrease volume.

volume_up(*value=None*)

Increase volume.

6.21 QTEMirror

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

class qtile_extras.widget.**QTEMirror**(**args*, ***kwargs*)

A modified version of Qtile's Mirror widget.

The only difference is to ensure mirrored widgets are sized correctly.

..important:

```
The mirror will also reflect any decorations of the original widget. Therefore,
if you need different decoration behaviour, you must create a new instance of the
widget.
```

This widget should not be created directly by users.

6.22 ScriptExit

class qtile_extras.widget.**ScriptExit**(**args*, ***kwargs*)

An updated version of Qtile's QuickExit widget.

Takes an additional argument `exit_script` which will be run before qtile exits.

Supported bar orientations: horizontal and vertical

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

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

set_font(*font=UNSPECIFIED*, *fontsize=UNSPECIFIED*, *fontshadow=UNSPECIFIED*)

Change the font used by this widget. If font is None, the current font is used.

trigger()

6.23 SnapCast

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

class qtile_extras.widget.**SnapCast**(**args*, ***kwargs*)

A widget to run a snapclient instance in the background.

This is a work in progress. The plan is to add the ability for the client to change groups from widget.

Required Dependencies

This module requires the following third-party libraries: `requests`

Supported bar orientations: horizontal only

Fig. 12: Snapclient active running in background

key	default	description
active_colour	'ffffff'	Colour when client is active and connected to server
background	None	Widget background color
client_name	None	Client name (as recognised by server).
decorations	[]	Decorations for widgets
error_colour	'ffff00'	Colour when client has an error (check logs)
icon_size	None	Icon size. None = autofit.
inactive_colour	'999999'	Colour when client is inactive
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
options	''	Options to be passed to snapclient.
padding	2	Padding around icon (and text).
server_address	'localhost'	Name or IP address of server.
server_reconnect	15	Interval before retrying to find player on server after failed attempt
snapclient	'/usr/bin/snapclient'	Path to snapclient

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

toggle_state()

Toggle Snapcast on and off.

6.24 StatusNotifier

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

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

A modified version of the default Qtile StatusNotifier widget.

Added the ability to render context menus by right-clicking on the icon.

Required Dependencies

This module requires the following third-party libraries: `dbus-next`, `xdg`

Supported bar orientations: horizontal and vertical

key	default	description
background	None	Widget background color
decorations	[]	Decorations for widgets
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_size	16	Icon width
icon_theme	None	Name of theme to use for app icons
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	200	Context menu width
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
opacity	1	Menu opactity
padding	3	Padding between icons
separator_colour	'555555'	Colour of menu separator
show_menu_icons	True	Show icons in context menu

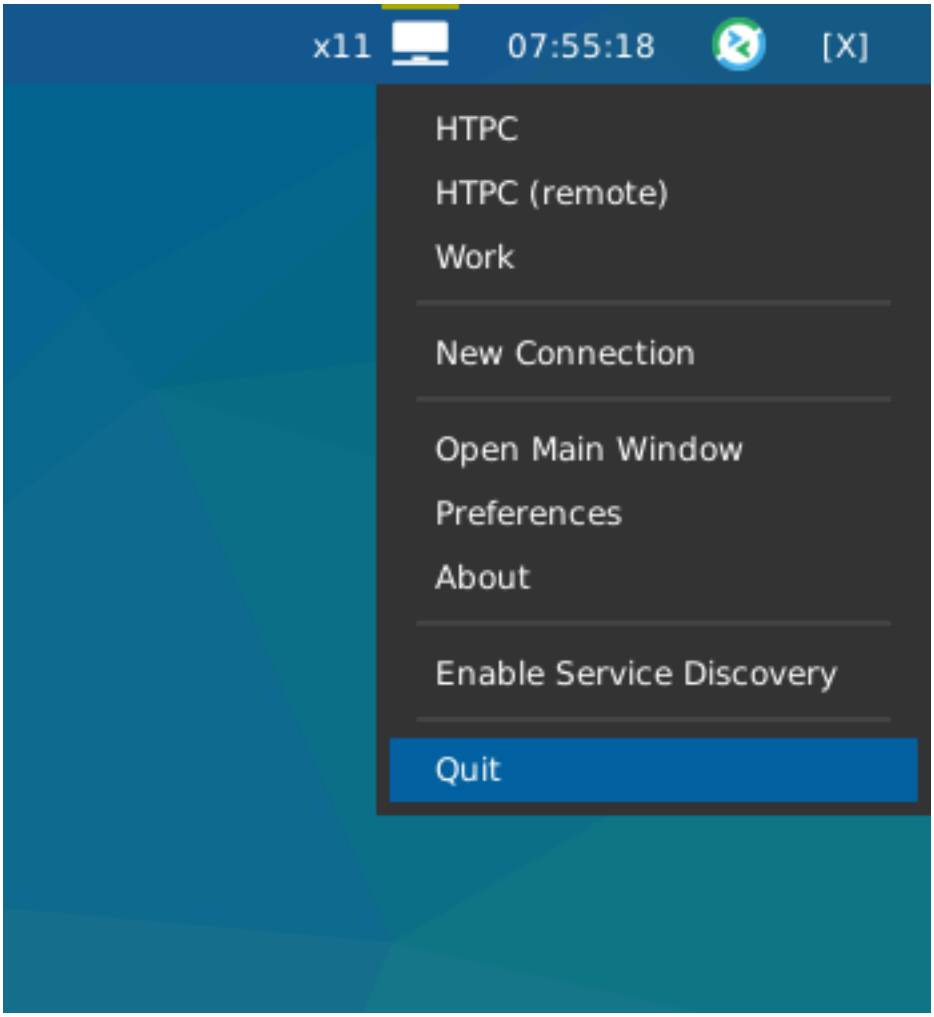


Fig. 13: Widget showing Remmina icon and context menu.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.25 StravaWidget

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

class qtile_extras.widget.**StravaWidget**(**args*, ***kwargs*)

This module provides a simple widget showing some Strava stats.

The widget text can be customised using the following keys:

prefix	suffix	value
C		Current month
Y		Calendar year
A		All time
	D	Distance
	C	Count
	T	Time
	P	Pace
	N	Name
	A	Date

For example, the default text `{CA:%b} {CD:.1f}km` displays the current date in abbreviated month name format and the distance run that month: "Aug 143.1km".

Extended info is provided by clicking on the widget.

Note: You will need to follow the instuctions at https://developers.strava.com/ to create a new app and authorise it.

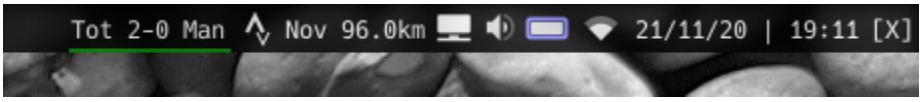
Your id and secret should be put in a json file called `auth.json` in `~/.cache/stravawidget`

The token file generated by the authorisation process, strava.json, should also be placed in the same folder.

Required Dependencies

This module requires the following third-party libraries: `stravalib`, `pint`

Supported bar orientations: horizontal only



key	default	description
`background`	`None`	Widget background color
`decorations`	`[]`	Decorations for widgets
`font`	`'sans'`	Default font
`fontsize`	`None`	Font size
`foreground`	`'ffffff'`	Text colour
`margin`	`3`	Margin inside the box
`margin_x`	`None`	X Margin. Overrides 'margin' if set
`margin_y`	`None`	Y Margin. Overrides 'margin' if set
`mouse_callbacks`	`{}`	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
`popup_display_ti`	`15`	Time to display extended info
`refresh_interval`	`1800`	Time to update data
`startup_delay`	`10`	Time before sending first web request
`text`	`'{CA:%b} {CD:.1f}km'`	Widget text
`warning_colour`	`'aaaa00'`	Highlight when there is an error.

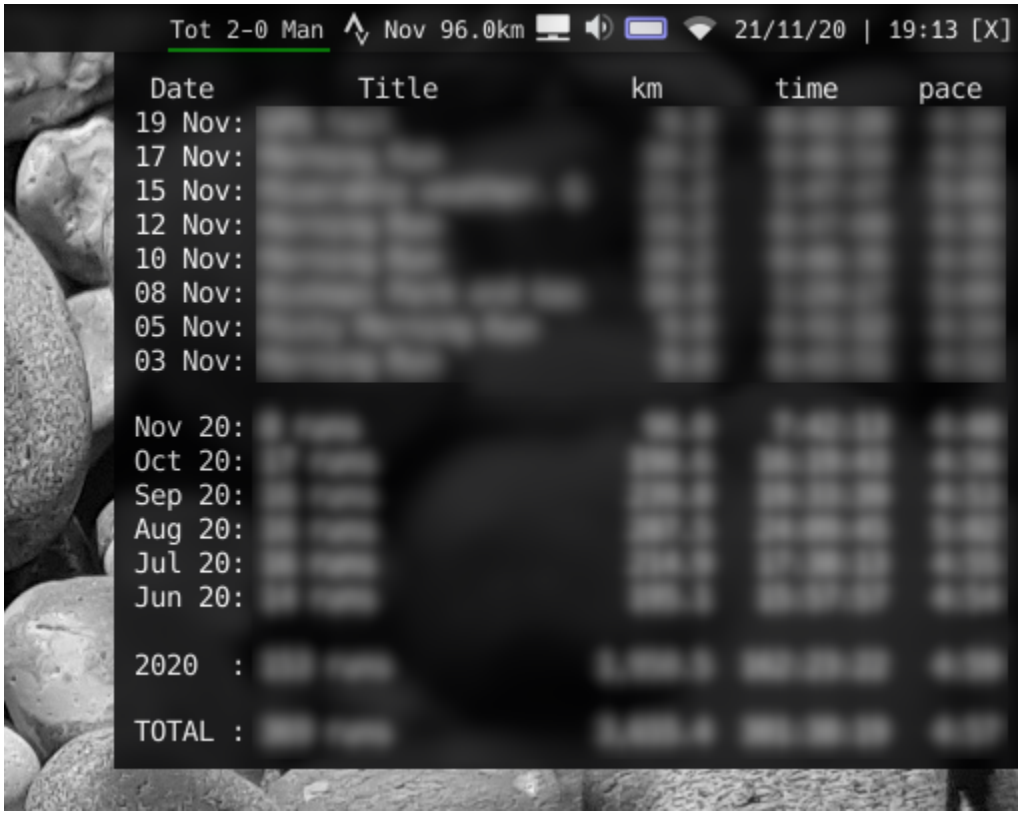


Fig. 14: Extended info. I've blurred out details of my runs for privacy reasons.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.26 SwapGraph

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

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

6.27 Syncthing

class qtile_extras.widget.**Syncthing**(**args*, ***kwargs*)

A widget to show the sync status of a Syncthing server.

By default, the widget displays an icon in the bar which is grey when no syncing is occurring but changes to white when syncing starts.

The widget can be configured to monitor a specific device or folder. By default, it monitors the local device at the `server` address.

Note: there is no verification of SSL certificates when connecting to the host. If this is a problem for you, please start an issue on the github page.

Required Dependencies

This module requires the following third-party libraries: `requests`

Supported bar orientations: horizontal only

Available hooks:

- st_sync_started
- st_sync_stopped

key	default	description
api_key	None	API key for the Syncthing server instance.
background	None	Widget background color
bar_background	None	Colour of bar background.
bar_colour	'008888'	Bar colour.
bar_height	10	Height of sync progress bar.
bar_text	''	Text to show over bar
bar_text_font	None	Font to use for bar text
bar_text_fontsiz	None	Fontsize for bar text
bar_text_foregro	'ffffff'	Colour for bar text
bar_text_format	'{percentage:.0%}'	Format string to display text on progress bar.
bar_width	75	Width of bar.
decorations	[]	Decorations for widgets
filter	{}	Limit the status check to a specific folder or device. Takes a dictionary where the key is either 'folder' or 'device' and the value is the appropriate ID. An empty 'folder' will aggregate all folders. An empty 'device' will use the local device. Default (empty dictionary) aggregates all folders on local device.
hide_on_idle	True	Hide widget if no sync in progress.
icon_colour_erro	'ffff00'	Colour for Syncthing logo when there's an error.
icon_colour_idle	'999999'	Colour for Syncthing logo when idle.
icon_colour_sync	'ffffff'	Colour for Syncthing logo when syncing.
icon_size	None	Icon size. None = autofit.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	2	Padding around icon.
server	'http://localhost:8384'	Syncthing API server.
show_bar	False	Show progress bar when syncing.
show_icon	True	Show icon.
update_interval	5	Time before updating status.
update_interval_	1	Time before updating while syncing.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.28 Systray

Note: This class has just been modified to enable compatibility with features provided by qtile-extras. No new functionality has been added.

class qtile_extras.widget.**Systray**(**args*, ***kwargs*)

A modified version of Qtile's Systray widget.

The only difference is to improve behaviour of the icon background when using `RectDecoration` decorations.

This widget does not and will not fix the issue with icons having a transparent background when displaying on a (semi-)transparent bar.

6.29 TVHWidget

class qtile_extras.widget.**TVHWidget**(**args*, ***kwargs*)

A widget to show whether a TVHeadend server is currently recording or not.

The widget will also show a popup displaying upcoming recordings.

When the server is recording a red line will be shown under the icon. If there's an error, a yellow line will show above the icon (and check the logs).

NB if you use a username and password, these are stored in plain text. You may therefore wish to create an unprivileged user account in TVHeadend that only has access to scheduled recordings data.

Required Dependencies

This module requires the following third-party libraries: `requests`

Supported bar orientations: horizontal only

Available hooks:

- tvh_recording_started
- tvh_recording_ended

key	default	description
auth	None	Auth details for accessing tvh. Can be None, tuple of (username, password).
auth_type	'basic'	HTTP authentication type: 'digest' or 'basic'
background	None	Widget background color
decorations	[]	Decorations for widgets
hide_duplicates	True	Remove duplicate recordings from list of upcoming recordings.
host	'http://localhost:9981/api'	TVHeadend server address
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
popup_display_ti	10	Seconds to show recordings.
popup_font	'monospace'	Font to use for displaying upcoming recordings. A monospace font is recommended
popup_format	'{start:%a %d %b %H:%M}: {title:.40}'	Upcoming recording text.
popup_opacity	0.8	Opacity for popup window.
popup_padding	10	Padding for popup window.
recording_colour	'bb0000'	Highlight when TVHeadend is recording
refresh_interval	30	Time to update data
startup_delay	5	Time before sending first web request
tvh_timeout	5	Seconds before timeout for timeout request
upcoming_recordi	'/dvr/entry/grid_upcoming'	API point for retrieving data on upcoming recordings.
warning_colour	'aaaa00'	Highlight when there is an error.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.30 UPowerWidget

class qtile_extras.widget.**UPowerWidget**(**args*, ***kwargs*)

A graphical widget to display laptop battery level.

The widget uses dbus to read the battery information from the UPower interface.

The widget will display one icon for each battery found or users can specify the name of the battery if they only wish to display one.

Clicking on the widget will display the battery level and the time to empty/full.

All colours can be customised as well as low/critical percentage levels.

Required Dependencies

This module requires the following third-party libraries: `dbus-next`

Supported bar orientations: horizontal only

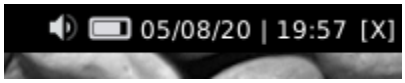


Fig. 15: Normal

Fig. 16: Low

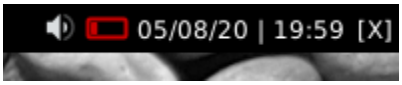


Fig. 17: Critical

Available hooks:

- up_power_connected

Fig. 18: Charging

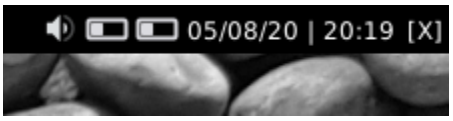


Fig. 19: Multiple batteries

- up_power_disconnected
- up_battery_full
- up_battery_low
- up_battery_critical

key	default	description
background	None	Widget background color
battery_height	10	Height of battery icon
battery_name	None	Battery name. None = all batteries
battery_width	20	Size of battery icon
border_charge_co	'8888ff'	Border colour when charging.
border_colour	'dbdbe0'	Border colour when discharging.
border_critical_	'cc0000'	Border colour when battery low.
decorations	[]	Decorations for widgets
fill_charge	None	Override fill colour when charging
fill_critical	'cc0000'	Fill when critically low
fill_low	'aa00aa'	Fill colour when battery low
fill_normal	'dbdbe0'	Fill when normal
font	'sans'	Default font
fontsize	None	Font size
foreground	'ffffff'	Font colour for information text
margin	2	Margin on sides of widget
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
percentage_criti	0.1	Critical level threshold.
percentage_low	0.2	Low level threshold.
spacing	5	Space between batteries
text_charging	'({percentage:.0f}%) {ttf} until fully charged'	Text to display when charging.
text_discharging	'({percentage:.0f}%) {tte} until empty'	Text to display when on battery.
text_displaytime	5	Time for text to remain before hiding

Fig. 20: Showing text

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.31 UnitStatus

class qtile_extras.widget.**UnitStatus**(**args*, ***kwargs*)

UnitStatus is a basic widget for Qtile which shows the current status of systemd units.

It may not be particular useful for you and was primarily written as an exercise to familiarise myself with writing Qtile widgets and interacting with d-bus.

The widget is incredibly basic. It subscribes to the systemd d-bus interface, finds the relevant service and displays an icon based on the current status. The widget listens for announced changes to the service and updates the icon accordingly.

Required Dependencies

This module requires the following third-party libraries: `dbus-next`

Supported bar orientations: horizontal only

key	default	description
background	None	Widget background color
bus_name	'system'	Which bus to use. Accepts 'system' or 'session'.
colour_active	'00ff00'	Colour for active indicator
colour_dead	'666666'	Colour for dead indicator
colour_failed	'ff0000'	Colour for active indicator
colour_inactive	'ffffff'	Colour for active indicator
decorations	[]	Decorations for widgets
font	'sans'	Default font
fontsize	None	Font size
foreground	'ffffff'	Font colour
indicator_size	10	Size of indicator (None = up to margin)
label	'NM'	Short text to display next to indicator.
margin	3	Margin inside the box
margin_x	None	X Margin. Overrides 'margin' if set
margin_y	None	Y Margin. Overrides 'margin' if set
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
padding	3	Padding inside the box
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
state_map	{'activating': ('colour_active' 'colour_inactive 'active': ('colour_active' 'colour_active') 'deactivating': ('colour_inactiv 'colour_active') 'dead': ('colour_dead', 'colour_dead'), 'failed': ('colour_failed' 'colour_failed') 'inactive': ('colour_inactiv 'colour_inactive 'not-found': ('colour_inactiv 'colour_failed')	Map of indicator colours (border, fill)
unitname	'NetworkManager.service'	Name of systemd unit.

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.32 Visualiser

Warning: This class has been marked as experimental.

The widget may behave unexpectedly, have missing features and will probably crash at some point!

Feedback on any issues would be appreciated.

class qtile_extras.widget.**Visualiser**(**args*, ***kwargs*)

A widget to draw an audio visualiser in your bar.

The widget requires cava to be installed. This may also be packaged by your distro.

cava is configured through the widget. Currently, you can set the number of bars and the framerate.

Warning: Rendering the visualiser directly in qtile's bar is almost certainly not an efficient way to have a visualiser in your setup. You should therefore be aware that this widget uses more processing power than other widgets so you may see CPU usage increase when using this. However, if the CPU usage continues to increase the longer you use the widget then that is likely to be a bug and should be reported!

Supported bar orientations: horizontal only

Fig. 21: Default config.

key	default	description
autostart	True	Start visualiser automatically
background	None	Widget background color
bar_colour	'#ffffff'	Colour of visualiser bars
bar_height	20	Height of visualiser bars
bars	8	Number of bars
cava_path	None	Path to cava. Set if file is not in your PATH.
cava_pipe	'/tmp/cava.pipe'	Pipe for cava's output
channels	'mono'	Visual channels. 'mono' or 'stereo'.
decorations	[]	Decorations for widgets
framerate	25	Cava sampling rate.
hide	True	Hide the visualiser when not active
invert	False	When True, bars will draw from the top down
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
spacing	2	Space between bars
width	100	Widget width

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

start()

Start the visualiser.

stop()

Stop this visualiser.

toggle()

Toggle visualiser state.

6.33 Visualizer

> **Warning:** This class has been marked as experimental.
>
> The widget may behave unexpectedly, have missing features and will probably crash at some point!
>
> Feedback on any issues would be appreciated.

qtile_extras.widget.**Visualizer**

alias of *Visualiser*

6.34 WiFiIcon

class qtile_extras.widget.**WiFiIcon**(**args*, ***kwargs*)

An simple graphical widget that shows WiFi status.

Left-clicking the widget will show the name of the network.

The widget can also periodically poll an external IP address to check whether the device is connected to the internet. To enable this, you need to set the *check_connection_interval*.

Required Dependencies

This module requires the following third-party libraries: `iwlib`

Supported bar orientations: horizontal only

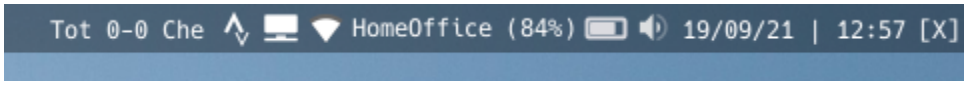


Fig. 22: Additional detail is visible when clicking on icon

key	default	description
active_colour	'ffffff'	Colour for wifi strength.
background	None	Widget background color
check_connection	0	Interval to check if device connected to internet (0 to disable)
decorations	[]	Decorations for widgets
disconnected_col	'aa0000'	Colour when device has no internet connection
expanded_timeout	5	Time in secs for expanded information to display when clicking on icon.
font	'sans'	Default font
fontsize	None	Font size
foreground	'ffffff'	Font colour for information text
inactive_colour	'666666'	Colour for wifi background.
interface	'wlan0'	Name of wifi interface.
internet_check_h	'8.8.8.8'	IP adddress to check for internet connection
internet_check_p	53	Port to check for internet connection
internet_check_t	5	Period before internet check times out and widget reports no internet connection.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and `lazy` calls.
padding	3	Padding inside the box
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
show_ssid	False	Show SSID and signal strength.
update_interval	1	Polling interval in secs.
wifi_arc	75	Angle of arc in degrees.
wifi_rectangle_w	5	Width of rectangle in pixels.
wifi_shape	'arc'	'arc' or 'rectangle'

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

hide()

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

show_text()

6.35 WordClock

class qtile_extras.widget.**WordClock**(**args*, ***kwargs*)

A widget to draw a word clock to the screen.

This is not a traditional widget in that you will not see anything displayed in your bar. The widget works in the background and updates the screen wallpaper when required. However, having this as a widget provides an easy way for users to install and configure the clock.

The clocks are currently designed to update on 5 minute intervals "five past" -> "ten past" etc. This may be changed in the future.

Custom layouts can be added by referring to the instructions in `qtile_extras/resources/wordclock/english.py`.

Supported languages

Available languages: `Dutch`, `English`, `Finnish`, `French`, `Portuguese`, `Spanish`, `Swedish`

Supported bar orientations: horizontal and vertical

key	default	description
active	'00AAAA'	Colour for active characters
background	'000000'	Background colour.
cache	'~/.cache/qtile-extras'	Location to store wallpaper
decorations	[]	Decorations for widgets
font	'sans'	Font for text
fontsize	70	Font size for letters
inactive	'202020'	Colour for inactive characters
language	'english'	Display language. Choose from 'dutch', 'english', 'finnish', 'french', 'portuguese', 'spanish', 'swedish'.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts functions and lazy calls.
update_interval	1	Interval to check time

commands() → list[str]

Returns a list of possible commands for this object

Used by __qsh__ for command completion and online help

doc(*name*) → str

Returns the documentation for a specified command name

Used by __qsh__ to provide online help.

eval(*code: str*) → tuple[bool, str | None]

Evaluates code in the same context as this function

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

function(*function*, **args*, ***kwargs*) → None

Call a function with current object as argument

info()

Info for this object.

items(*name: str*) → tuple[bool, list[str | int] | None]

Build a list of contained items for the given item class.

Exposing this allows __qsh__ to navigate the command graph.

Returns a tuple *(root, items)* for the specified item class, where:

root: True if this class accepts a "naked" specification without an item seletion (e.g. "layout" defaults to current layout), and False if it does not (e.g. no default "widget").

items: a list of contained items

6.36 Mixins

6.36.1 ConnectionCheckMixin

class qtile_extras.widget.mixins.**ConnectionCheckMixin**

Mixin to periodically check for internet connection and set the `self.is_connected` flag depending on status.

Your code should include the following lines to use the mixin.

```
class MyInternetWidget(ConnectionCheckMixin):
    def __init__(self):
        self.add_defaults(ConnectionCheckMixin.defaults)
        ConnectionCheckMixin.__init__(self)

    def _configure(self, qtile, bar):
        ConnectionCheckMixin._configure(self)
```

key	default	description
check_connection	0	Interval to check if device connected to internet (0 to disable)
disconnected_col	'aa0000'	Colour when device has no internet connection
internet_check_h	'8.8.8.8'	IP adddress to check for internet connection
internet_check_p	53	Port to check for internet connection
internet_check_t	5	Period before internet check times out and widget reports no internet connection.

6.36.2 DbusMenuMixin

class qtile_extras.widget.mixins.**DbusMenuMixin**(***config*)

Builds a menu from `qtile_extras.resources.dbusmenu.DBusMenuItem` objects.

Should be used where a widget is accessing menu data over DBus.

When calling `qtile_extras.resources.dbusmenu.DBusMenu.get_menu`, the callback should be set to the widget's `display_menu` method.

key	default	description
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_theme	None	Icon theme for DBus menu items
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	200	Context menu width
opacity	1	Menu opactity
separator_colour	'555555'	Colour of menu separator
show_menu_icons	True	Show icons in context menu

6.36.3 ExtendedPopupMixin

class qtile_extras.widget.mixins.**ExtendedPopupMixin**(***kwargs*)

Mixin that provides the ability for a widget to display extended detail in popups via the Popup toolkit.

It is not mandatory for widgets to use this if they want to use the toolkit. However, the mixin provides some standard variable and method names.

self.extended_popup	the popup instance or None
self._popup_hide_tim	the current timer object for hiding the popup or None
self.has_popup	property that returns True is popup is defined and not killed
self.update_popup()	method to call to update popup contents. Should not be overriden as it calls self._update_popup (see below) but only if self.has_popup is True
self._update_popup()	method that actually updates the contents. This will raise a NotImplementedError if called without being overriden.
self._set_popup_time	sets the timer to kill the popup.
self._kill_popup()	kills the popup
self.show_popup()	displays the popup. Is also exposed to command interface so can be used in lazy calls etc.

key	default	description
popup_hide_timeo	0	Number of seconds before popup is hidden (0 to disable).
popup_layout	None	The popup layout definition
popup_show_args	{'centered': True}	Arguments to be passed to popup.show()

6.36.4 GraphicalWifiMixin

class qtile_extras.widget.mixins.**GraphicalWifiMixin**

Provides the ability to draw a graphical representation of wifi signal strength.

To use the mixin, your code needs to include the following:

```
class MyGraphicalInternetWidget(GraphicalWifiMixin):
    def __init__(self):
        self.add_defaults(GraphicalWifiMixin.defaults)
        GraphicalWifiMixin.__init__(self)

    def _configure(self, qtile, bar):
        ... # other configuration lines here

        self.set_wifi_sizes()

    def draw(self):
        # To draw the icon you need the following parameters:
        # - percentage: a value between 0 and 1
        # - foreground: the colour of the indicator
        # - background: the colour of the indicator background
        self.draw_wifi(percentage=percentage, foreground=foreground,␣
↪background=background)
```

Note: This mixin does not set the width of your widget but does provide a `self.wifi_width` attribute which can be used for this purpose.

key	default	description
wifi_arc	75	Angle of arc in degrees.
wifi_rectangle_w	5	Width of rectangle in pixels.
wifi_shape	'arc'	'arc' or 'rectangle'

6.36.5 MenuMixin

class qtile_extras.widget.mixins.**MenuMixin**(***config*)

Provides the relevant settings to help configure a context menu to be displayed by the widget.

The use of the mixin ensures that all menus use the same property names which allows users to theme menus more easily e.g. by setting values in `widget_defaults`.

key	default	description
hide_after	0.5	Time in seconds before hiding menu atfer mouse leave
highlight_colour	'0060A0'	Colour of highlight for menu items (None for no highlight)
highlight_radius	0	Radius for menu highlight
icon_theme	None	Icon theme for DBus menu items
menu_background	'333333'	Background colour for menu
menu_border	'111111'	Menu border colour
menu_border_widt	0	Width of menu border
menu_font	'sans'	Font for menu text
menu_fontsize	12	Font size for menu text
menu_foreground	'ffffff'	Font colour for menu text
menu_foreground_	'aaaaaa'	Font colour for disabled menu items
menu_foreground_	None	Font colour for highlighted item (None to use menu_foreground value)
menu_icon_size	12	Size of icons in menu (where available)
menu_offset_x	0	Fine tune x position of menu
menu_offset_y	0	Fine tune y position of menu
menu_row_height	None	Height of menu row (NB text entries are 2 rows tall, separators are 1 row tall.) "None" will attempt to calculate height based on font size.
menu_width	200	Context menu width
opacity	1	Menu opactity
separator_colour	'555555'	Colour of menu separator
show_menu_icons	True	Show icons in context menu

6.36.6 ProgressBarMixin

class qtile_extras.widget.mixins.**ProgressBarMixin**(***kwargs*)

Mixin to allow widgets to display progress bars.

Bar is drawn based on a `bar_value` between 0.0 and 1.0 inclusive.

To use it, subclass and add this to `__init__`:

```
ProgressBarMixin.__init__(self, **kwargs)
self.add_defaults(ProgressBarMixin.defaults)
```

To draw the bar, you need to call `self.bar_draw()`. The method take a number of optional parameters. Where these are not set in the method call then the instance version i.e. `self.parameter_name` will be used insted.

`bar.draw` optional parameters:

- `x_offset` (default 0): horizontal positioning of the bar
- `bar_colour`: colour of the bar
- `bar_background`: colour drawn behind the bar (i.e. to show extent of bar)

- `bar_text`: text to draw on bar,
- `bar_text_foreground`: text colour,
- `bar_value`: percentage of bar to fill

Note: The widget should ensure that its width is sufficient to display the bar (the `bar_width` property is relevant here).

key	default	description
bar_background	None	Colour of bar background.
bar_colour	'00ffff'	Colour of bar (NB this setting may be overridden by other widget settings).
bar_height	None	Height of bar (None = full bar height).
bar_text	''	Text to show over bar
bar_text_font	None	Font to use for bar text
bar_text_fontsiz	None	Fontsize for bar text
bar_text_foregro	'ffffff'	Colour for bar text
bar_width	75	Width of bar.

6.36.7 TooltipMixin

class qtile_extras.widget.mixins.**TooltipMixin**(***kwargs*)

Mixin that provides a tooltip for widgets.

To use it, subclass and add this to `__init__`:

```
TooltipMixin.__init__(self, **kwargs)
self.add_defaults(TooltipMixin.defaults)
```

Widgets should set `self.tooltip_text` to change display text.

key	default	description
tooltip_backgrou	'#000000'	Background colour for tooltip
tooltip_color	'#ffffff'	Font colur for tooltop
tooltip_delay	1	Time in seconds before tooltip displayed
tooltip_font	'sans'	Font colour for tooltop
tooltip_fontsize	12	Font size for tooltop
tooltip_padding	4	int for all sides or list for [top/bottom, left/right]

CHAPTER

SEVEN

HOOKS

subscribe.**ghn_new_notification**()

GithubNotifications widget.

Fired when there is a new notification.

Note: the hook will only be fired whenever the widget polls.

```
from libqtile import qtile
from libqtile.utils import send_notification

import qtile_extras.hook

@qtile_extras.hook.subscribe.ghn_new_notification
def ghn_notification():
    qtile.spawn("ffplay ding.wav")
```

subscribe.**lfs_goal_scored**()

LiveFootballScores widget.

Fired when the score in a match changes.

Hooked function should receive one argument which is the `FootballMatch` object for the relevant match.

Note: as the widget polls all matches at the same time, you may find that the hook is fired multiple times in quick succession. Handling multiple hooks is left to the user to manage.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.lfs_goal_scored
def goal(match):
    if "Arsenal" in (match.home_team, match.away_team):
      qtile.spawn("ffplay goal.wav")
```

subscribe.**lfs_red_card**()

LiveFootballScores widget.

Fired when a red card is issued in a match.

Hooked function should receive one argument which is the `FootballMatch` object for the relevant match.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.lfs_red_card
def red_card(match):
    if "Arsenal" in (match.home_team, match.away_team):
        qtile.spawn("ffplay off.wav")
```

subscribe.**lfs_status_change**()

LiveFootballScores widget.

Fired when the match status changes (i.e. kick-off, half time etc.).

Hooked function should receive one argument which is the `FootballMatch` object for the relevant match.

Note: as the widget polls all matches at the same time, you may find that the hook is fired multiple times in quick succession. Handling multiple hooks is left to the user to manage.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.lfs_status_change
def status(match):
    if match.is_finished and "Arsenal" in (match.home_team, match.away_team):
        qtile.spawn("ffplay whistle.wav")
```

subscribe.**mpris_new_track**()

Mpris2 widget.

Fired when a track changes. Receives a dict of the new metadata.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.mpris_new_track
def new_track(metadata):
    if metadata["xesam:title"] == "Never Gonna Give You Up":
        qtile.spawn("max_volume.sh")
```

subscribe.**mpris_status_change**()

Mpris2 widget.

Fired when the playback status changes. Receives a string containing the new status.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.mpris_status_change
def new_track(status):
    if status == "Stopped":
        qtile.spawn("mute.sh")
```

(continues on next page)

(continued from previous page)

```
    else:
        qtile.spawn("unmute.sh")
```

subscribe.**st_sync_started**()

Syncthing widget.

Fired when a sync starts.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.st_sync_started
def sync_start():
    qtile.spawn("ffplay start.wav")
```

subscribe.**st_sync_stopped**()

Syncthing widget.

Fired when a sync stops.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.st_sync_stopped
def sync_stop():
    qtile.spawn("ffplay complete.wav")
```

subscribe.**tvh_recording_ended**()

TVHeadend widget.

Fired when a recording ends.

Hooked function should receive one argument which is the name of the program that was recorded.

```
from libqtile.utils import send_notification

import qtile_extras.hook

@qtile_extras.hook.subscribe.tvh_recording_ended
def stop_recording(prog):
    send_notification("Recording Ended", prog)
```

subscribe.**tvh_recording_started**()

TVHeadend widget.

Fired when a recording starts.

Hooked function should receive one argument which is the name of the program being recorded.

```
from libqtile.utils import send_notification

import qtile_extras.hook
```

(continues on next page)

(continued from previous page)

```
@qtile_extras.hook.subscribe.tvh_recording_started
def start_recording(prog):
    send_notification("Recording Started", prog)
```

subscribe.**up_battery_critical**()

UPowerWidget.

Fired when a battery is critically low.

```
from libqtile.utils import send_notification

import qtile_extras.hook

@qtile_extras.hook.subscribe.up_battery_critical
def battery_critical(battery_name):
    send_notification(battery_name, "Battery is critically low. Plug in power␣
↪supply.")
```

subscribe.**up_battery_full**()

UPowerWidget.

Fired when a battery is fully charged.

```
from libqtile.utils import send_notification

import qtile_extras.hook

@qtile_extras.hook.subscribe.up_battery_full
def battery_full(battery_name):
    send_notification(battery_name, "Battery is fully charged.")
```

subscribe.**up_battery_low**()

UPowerWidget.

Fired when a battery reaches low threshold.

```
from libqtile.utils import send_notification

import qtile_extras.hook

@qtile_extras.hook.subscribe.up_battery_low
def battery_low(battery_name):
    send_notification(battery_name, "Battery is running low.")
```

subscribe.**up_power_connected**()

UPowerWidget.

Fired when a power supply is connected.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.up_power_connected
```

(continues on next page)

(continued from previous page)

```
def plugged_in():
    qtile.spawn("ffplay power_on.wav")
```

subscribe.**up_power_disconnected**()

UPowerWidget.

Fired when a power supply is disconnected.

```
from libqtile import qtile

import qtile_extras.hook

@qtile_extras.hook.subscribe.up_power_disconnected
def unplugged():
    qtile.spawn("ffplay power_off.wav")
```

CHAPTER

EIGHT

POPUP TOOLKIT

8.1 Popup Toolkit Layouts

Layouts are the container that houses all the controls. In order to give the greatest flexibility to users, there are a number of different layouts available in the toolkit.

8.1.1 PopupAbsoluteLayout

class qtile_extras.popup.toolkit.**PopupAbsoluteLayout**(*qtile: Qtile | None = None*, ***config*)

The absolute layout is the simplest layout of all. Controls are placed based on the following parameters:

```
`pos_x`, `pos_y`: top left corner
`width`, `height`: size of control
```

No further adjustments are made to the controls.

Note: the layout currently ignores the `margin` attribute i.e. a control placed at (0,0) will display there even if a margin is defined.

key	default	description
background	'000000'	Popup background colour
border	'111111'	Border colour for popup
border_width	0	Popup border width
close_on_click	True	Hide the popup when control is clicked
controls	[]	Controls to display
height	200	Height of tooltip
hide_interval	0.5	Timeout after mouse leaves popup before popup is lilled
hide_on_mouse_le	False	Hide the popup if the mouse pointer leaves the popup
hide_on_timeout	0	Timeout before popup closes (0 = disabled). Useful for notifications
initial_focus	0	Index of control to be focused at startup.
keyboard_navigat	True	Whether popup controls can be navigated by keys
keymap	{'close': ['Escape'], 'down': ['Down', 'k'], 'left': ['Left', 'h'], 'right': ['Right', 'l'], 'select': ['Return', 'space'], 'step': ['Tab'], 'up': ['Up', 'j']}	Keyboard controls. NB Navigation logic is very rudimentary. The popup will try to select the nearest control in the direction pressed but some controls may be inaccessible. In that scenario, use the mouse or *Tab* to cycle through controls.
margin	5	Margin around edge of tooltip
opacity	1	Popup window opacity. 'None' inherits bar opacity
width	200	Width of tooltip

8.1.2 PopupGridLayout

class qtile_extras.popup.toolkit.**PopupGridLayout**(*qtile*, ***config*)

The grid layout should be familiar to users who have used Tkinter.

In addition to the *width* and *height* attributes, the grid layout also requires *rows* and *cols* to define the grid. Grid cells are evenly sized.a

Controls can then be placed in the grid via the *row*, *col*, *row_span* and *col_span* parameters.

For example:

```
PopupGridLayout(rows=6, cols=6, controls=[
    PopupImage(filename="A",row=0, col=2, row_span=2, col_span=2),
    PopupImage(filename="B",row=2, col=2, row_span=2, col_span=2),
    PopupImage(filename="C",row=3, col=1),
    PopupImage(filename="D",row=3, col=4),
    PopupText(row=4,col_span=6),
])
```

would result in a tooltip looking like:

```
------------------------
|   |   |       |   |   |
---------   A   ---------
|   |   |       |   |   |
------------------------
|   |   |       |   |   |
---------   B   ---------
|   | C |       | D |   |
------------------------
|         TEXT           |
------------------------
|   |   |   |   |   |   |
------------------------
```

row and col are both zero-indexed.

key	default	description
background	'000000'	Popup background colour
border	'111111'	Border colour for popup
border_width	0	Popup border width
close_on_click	True	Hide the popup when control is clicked
cols	2	Number of columns in grid
controls	[]	Controls to display
height	200	Height of tooltip
hide_interval	0.5	Timeout after mouse leaves popup before popup is lilled
hide_on_mouse_le	False	Hide the popup if the mouse pointer leaves the popup
hide_on_timeout	0	Timeout before popup closes (0 = disabled). Useful for notifications
initial_focus	0	Index of control to be focused at startup.
keyboard_navigat	True	Whether popup controls can be navigated by keys
keymap	{'close': ['Escape'], 'down': ['Down', 'k'], 'left': ['Left', 'h'], 'right': ['Right', 'l'], 'select': ['Return', 'space'], 'step': ['Tab'], 'up': ['Up', 'j']}	Keyboard controls. NB Navigation logic is very rudimentary. The popup will try to select the nearest control in the direction pressed but some controls may be inaccessible. In that scenario, use the mouse or *Tab* to cycle through controls.
margin	5	Margin around edge of tooltip
opacity	1	Popup window opacity. 'None' inherits bar opacity
rows	2	Number of rows in grid
width	200	Width of tooltip

8.1.3 PopupRelativeLayout

class qtile_extras.popup.toolkit.**PopupRelativeLayout**(*qtile: Qtile | None = None*, ***config*)

The relative layout positions controls based on a percentage of the parent tooltip's dimensions.

The positions are defined with the following parameters:

```
`pos_x`, `pos_y`: top left corner
`width`, `height`: size of control
```

All four of these parameters should be a value between 0 and 1. Values outside of this range will generate a warning in the log but will not raise an exception.

For example:

```
PopupRelativeLayout(rows=6, cols=6, controls=[
    PopupImage(filename="A",pos_x=0.1, pos_y=0.2, width=0.5, height=0.5)
])
```

Would result in a tooltip with dimensions of 200x200 (the default), with an image placed at (20, 40) with dimensions of (100, 100).

Note: images are not stretched but are, instead, centered within the rect.

key	default	description
background	'000000'	Popup background colour
border	'111111'	Border colour for popup
border_width	0	Popup border width
close_on_click	True	Hide the popup when control is clicked
controls	[]	Controls to display
height	200	Height of tooltip
hide_interval	0.5	Timeout after mouse leaves popup before popup is lilled
hide_on_mouse_le	False	Hide the popup if the mouse pointer leaves the popup
hide_on_timeout	0	Timeout before popup closes (0 = disabled). Useful for notifications
initial_focus	0	Index of control to be focused at startup.
keyboard_navigat	True	Whether popup controls can be navigated by keys
keymap	{'close': ['Escape'], 'down': ['Down', 'k'], 'left': ['Left', 'h'], 'right': ['Right', 'l'], 'select': ['Return', 'space'], 'step': ['Tab'], 'up': ['Up', 'j']}	Keyboard controls. NB Navigation logic is very rudimentary. The popup will try to select the nearest control in the direction pressed but some controls may be inaccessible. In that scenario, use the mouse or *Tab* to cycle through controls.
margin	5	Margin around edge of tooltip
opacity	1	Popup window opacity. 'None' inherits bar opacity
width	200	Width of tooltip

8.2 Popup Toolkit Controls

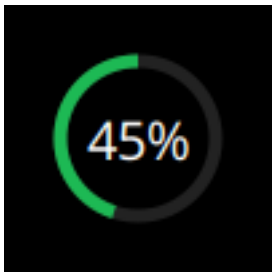
Controls are the building blocks for your popup. You can place text, images and progress bars and have each of these react to input events or display information dynamically.

8.2.1 PopupCircularProgress

class qtile_extras.popup.toolkit.**PopupCircularProgress**(*value=None*, ***config*)

Draws a circular progress bar.

`colour_below` should be used to set the colour of the progress bar while `colour_above` will draw a background ring.



Note: This is a special control that is designed to be used in conjunction with other controls. By default, `clip` is set to `True` which means that *only* the area that would be covered by the full circular progress bar is drawn i.e. there is no rectangular background to this control. The result is that any control underneath the circular progress bar is visible.

To place this control above another, it should be defined *after* that control. For example:

```
layout = PopupRelativeLayout(
    controls=[
        PopupImage(...),  # image to display beneath progress bar
        PopupCircularProgress(...)
    ]
)
```

key	default	description
background	None	Background colour for control
bar_border_colou	'ffffff'	Colour of border drawn around bar
bar_border_margi	0	Size of gap between border and bar
bar_border_size	0	Thickness of border around bar
bar_size	2	Thickness of bar
can_focus	'auto'	Whether or not control can be focussed. Focussed control will be highlighted if *highlight* attribute is set. Possible value are: True, False or 'auto' (which sets to True if a 'Button1' mouse_callback is set).
clip	True	When True the drawing area is limited to the circular progress bar. This allows the progress bar to be placed on top of other controls and still show their content.

continues on next page

Table 1 – continued from previous page

key	default	description
clockwise	True	Progress increases in a clockwise direction.
col	0	Column position (for grid layout)
col_span	1	Number of columns covered by control
colour_above	'#888888'	Colour for bar above value
colour_below	'#ffffff'	Colour for bar below value
end_margin	5	Gap between edge of control and ends of the bar/border
height	50	height of control
highlight	'#006666'	Highlight colour
highlight_border	2	Border width for focused controls
highlight_method	'block'	How to highlight focused control. Options are 'border' and 'block'.
highlight_radius	5	Corner radius for highlight
horizontal	True	Orientation. False = vertical
marker_colour	'#bbbbbb'	Colour of marker
marker_size	10	Size of marker
max_value	1.0	Maximum value
min_value	0	Minimum value
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts lazy objects.
name	None	A unique name for the control. Is only necessary if you wish to update the control's value via `popup.update_controls()`.
pos_x	0	x position of control
pos_y	0	y position of control
row	0	Row position (for grid layout)
row_span	1	Number of rows covered by control
start_angle	0	Starting angle (in degrees) for progress marker. 0 is 12 o'clock and angle increases in a clockwise direction.
width	50	width of control

8.2.2 PopupImage

class qtile_extras.popup.toolkit.**PopupImage**(***config*)

Control to display an image.

Image will be scaled (locked aspect ratio) to fit within the control rect. The image will also be centered vertically and horizontally.

key	default	description
background	None	Background colour for control
can_focus	'auto'	Whether or not control can be focussed. Focussed control will be highlighted if *highlight* attribute is set. Possible value are: True, False or 'auto' (which sets to True if a 'Button1' mouse_callback is set).
col	0	Column position (for grid layout)
col_span	1	Number of columns covered by control
colour	'ffffff'	Colour to use to fill image when using mask mode
filename	None	path to image file.
height	50	height of control
highlight	'#006666'	Highlight colour
highlight_border	2	Border width for focused controls
highlight_filena	None	path to image to be displayed when highlight method is 'image'
highlight_method	'block'	How to highlight focused control. Options are 'image', 'border', 'block' and 'mask'.
highlight_radius	5	Corner radius for highlight
mask	False	Use the image as a mask
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts lazy objects.
name	None	A unique name for the control. Is only necessary if you wish to update the control's value via `popup.update_controls()`.
pos_x	0	x position of control
pos_y	0	y position of control
row	0	Row position (for grid layout)
row_span	1	Number of rows covered by control
width	50	width of control

8.2.3 PopupSlider

class qtile_extras.popup.toolkit.**PopupSlider**(*value=None*, ***config*)

Control to display slider/progress bar.

Bar can be displayed horizontally (draws left-to-right) or vertically (bottom-to-top).

In addition, a border can be drawn around the bar using the `bar_border_colour/size/margin` parameters.

key	default	description
background	None	Background colour for control
bar_border_colou	'ffffff'	Colour of border drawn around bar
bar_border_margi	0	Size of gap between border and bar
bar_border_size	0	Thickness of border around bar
bar_size	2	Thickness of bar
can_focus	'auto'	Whether or not control can be focussed. Focussed control will be highlighted if *highlight* attribute is set. Possible value are: True, False or 'auto' (which sets to True if a 'Button1' mouse_callback is set).
col	0	Column position (for grid layout)
col_span	1	Number of columns covered by control
colour_above	'#888888'	Colour for bar above value
colour_below	'#ffffff'	Colour for bar below value
end_margin	5	Gap between edge of control and ends of the bar/border
height	50	height of control
highlight	'#006666'	Highlight colour
highlight_border	2	Border width for focused controls
highlight_method	'block'	How to highlight focused control. Options are 'border' and 'block'.
highlight_radius	5	Corner radius for highlight
horizontal	True	Orientation. False = vertical
marker_colour	'#bbbbbb'	Colour of marker
marker_size	10	Size of marker
max_value	1.0	Maximum value
min_value	0	Minimum value
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts lazy objects.
name	None	A unique name for the control. Is only necessary if you wish to update the control's value via `popup.update_controls()`.
pos_x	0	x position of control
pos_y	0	y position of control
row	0	Row position (for grid layout)
row_span	1	Number of rows covered by control
width	50	width of control

8.2.4 PopupText

class qtile_extras.popup.toolkit.**PopupText**(*text=''*, ***config*)

Simple control to display text.

key	default	description
background	None	Background colour for control
can_focus	'auto'	Whether or not control can be focussed. Focussed control will be highlighted if *highlight* attribute is set. Possible value are: True, False or 'auto' (which sets to True if a 'Button1' mouse_callback is set).
col	0	Column position (for grid layout)
col_span	1	Number of columns covered by control
font	'sans'	Font name
fontsize	12	Font size
foreground	'#ffffff'	Font colour
foreground_highl	None	Font colour when highlighted via *block* (None to use foreground value)
h_align	'left'	Text alignment: left, center or right.
height	50	height of control
highlight	'#006666'	Highlight colour
highlight_border	2	Border width for focused controls
highlight_method	'block'	Available options: 'border', 'block' or 'text'.
highlight_radius	5	Corner radius for highlight
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts lazy objects.
name	None	A unique name for the control. Is only necessary if you wish to update the control's value via `popup.update_controls()`.
pos_x	0	x position of control
pos_y	0	y position of control
row	0	Row position (for grid layout)
row_span	1	Number of rows covered by control
v_align	'middle'	Vertical alignment: top, middle or bottom.
width	50	width of control
wrap	False	Wrap text in layout

8.2.5 PopupWidget

class qtile_extras.popup.toolkit.**PopupWidget**(**_config_)

Control to display a Qtile widget in a Popup window.

Mouse clicks are passed on to the widgets.

Currently, widgets will be sized based on the dimensions of the control. This will override any width/stretching settings in thw widget.

key	default	description
background	None	Background colour for control
can_focus	'auto'	Whether or not control can be focussed. Focussed control will be highlighted if *highlight* attribute is set. Possible value are: True, False or 'auto' (which sets to True if a 'Button1' mouse_callback is set).
col	0	Column position (for grid layout)
col_span	1	Number of columns covered by control
height	50	height of control
highlight	'#006666'	Highlight colour
highlight_border	2	Border width for focused controls
highlight_method	'block'	How to highlight focused control. Options are 'border' and 'block'.
highlight_radius	5	Corner radius for highlight
horizontal	True	Widget is horizontal. False = vertical.
mouse_callbacks	{}	Dict of mouse button press callback functions. Accepts lazy objects.
name	None	A unique name for the control. Is only necessary if you wish to update the control's value via `popup.update_controls()`.
pos_x	0	x position of control
pos_y	0	y position of control
row	0	Row position (for grid layout)
row_span	1	Number of rows covered by control
vertical_left	True	If using vertical orientation, mimic bar on left hand side of screen (causes text to read from bottom to top).
widget	None	Widget instance.
width	50	width of control

CHAPTER

NINE

DECORATIONS

9.1 BorderDecoration

class qtile_extras.widget.decorations.**BorderDecoration**(***config*)

Widget decoration that draws a straight line on the widget border. Padding can be used to adjust the position of the border further.

Only one colour can be set but decorations can be layered to achieve multi-coloured effects.

Fig. 1: Stacked borders

key	default	description
border_width	2	Border width as int or list of ints [N E S W].
colour	'#000000'	Border colour
extrawidth	0	Add additional width to the end of the decoration
group	False	When set to True, the decoration will be applied as if the widgets were grouped. See documentation for more.
ignore_extrawidt	False	Ignores additional width added by decoration. Useful when stacking decorations on top of a PowerLineDecoration.
padding	0	Default padding
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set

9.2 PowerLineDecoration

class qtile_extras.widget.decorations.**PowerLineDecoration**(***config*)

Widget decoration that can be used to recreate the PowerLine style.

The advantages of the decoration are:

- No fonts needed
- The same decoration definition can be used for all widgets (the decoration works out which background and foreground colours to use)
- Different styles can be used by changing a few parameters of the decoration

The decoration works by adding the shape **after** the current widget. The decoration will also be hidden if a widget has zero width (i.e. is hidden).

The shape is drawn in area whose size is defined by the `size` parameter. This area is drawn after the widget but can be shifted back by using the `shift` parameter. Shifting too far will result in widget contents being drawn over the shape.

By default, the decoration will set colours based on the backgrounds of the adjoining widgets. The left-hand portion of the decoration is determined by the decorated widget, the right-hand portion comes from the next visible widget in the bar (or the bar background if the decorated widget is the last widget in the bar). Both colours can be overriden by using the *override_colour* and *override_next_colour* parameters.

The default behavious is to draw an arrow pointing right. To change the shape you can use pre-defined paths: "arrow_left", "arrow_right", "forward_slash", "back_slash", "rounded_left", "rounded_right" and "zig_zag". Alternatively, you can create a custom shape by defining a path. The format is a list of (x, y) tuples. x and y values should be between 0 and 1 to represent the relative position in the additional space created by the decoration. (0, 0) is the top left corner (on a horizontal widget) and (1, 1) is the bottom right corner. The first point in the list is the starting point and then a line will be drawn to each subsequent point. The path is then closed by returning to the first point. Finally, the shape is filled with the background of the current widget.

Note: The decoration currently only works on horizontal bars. The `padding_y` parameter can be used to adjust the vertical size of the decoration. However, note that this won't change the size of the widget's own background. If you want to do that, you can use the following:

```
powerline = {
    "decorations": [
        RectDecoration(use_widget_background=True, padding_y=5, filled=True,␣
↪radius=0),
        PowerLineDecoration(path="arrow_right", padding_y=5)
    ]
}
```

The RectDecoration has the same padding and will use the widget's `background` parameter as its own colour.

Example code:

```
from qtile_extras import widget
from qtile_extras.widget.decorations import PowerLineDecoration

powerline = {
    "decorations": [
        PowerLineDecoration()
    ]
}

screens = [
    Screen(
        top=Bar(
            [
                widget.CurrentLayoutIcon(background="000000", **powerline),
                widget.WindowName(background="222222", **powerline),
                widget.Clock(background="444444", **powerline),
                widget.QuickExit(background="666666")
            ],
```

(continues on next page)

(continued from previous page)

```
            30
        )
    )
]
```

The above code generates the following bar:

Fig. 2: path='arrow_right'

Fig. 3: path='rounded_left'

Fig. 4: path='rounded_right'

key	default	description
extrawidth	0	Add additional width to the end of the decoration
ignore_extrawidt	False	Ignores additional width added by decoration. Useful when stacking decorations on top of a PowerLineDecoration.
override_colour	None	Force background colour.
override_next_co	None	Force background colour for the next part of the decoration.
padding	0	Default padding
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
path	'arrow_left'	Shape of decoration. See docstring for more info.
shift	0	Number of pixels to shift the decoration back by.
size	15	Width of shape

Fig. 5: path='forward_slash'

Fig. 6: path='back_slash'

9.3 RectDecoration

class qtile_extras.widget.decorations.**RectDecoration**(***config*)

Widget decoration that draws a rectangle behind the widget contents.

Rectangles can be drawn as just the the outline or filled and the outline can be a different colour to fill. In addition, decorations can be layered to achieve more multi-coloured effects.

Curved corners can be obtained by setting the `radius` parameter.

To have the effect of multiple widgets using the same decoration (e.g. the curved ends appearing on the first and last widgets) use the `group=True` option.

The advantage of using the `group` option is that the group is dynamic meaning that it is drawn according to the widgets that are currently visible in the group. The group will adjust if a window becomes visible or hidden.

```
decoration_group = {
    "decorations": [
        RectDecoration(colour="#004040", radius=10, filled=True, padding_y=4,␣
↪group=True)
    ],
    "padding": 10,
}

screens = [
    Screen(
        top=Bar(
            [
                extrawidgets.CurrentLayout(**decoration_group),
                widget.Spacer(),
                extrawidgets.StatusNotifier(**decoration_group),
                extrawidgets.Mpris2(**decoration_group),
                extrawidgets.Clock(format="%H:%M:%S", **decoration_group),
                extrawidgets.ScriptExit(
                    default_text="[X]",
                    countdown_format="[{}]",
                    countdown_start=2,
                    **decoration_group
```

(continues on next page)

Fig. 7: path='zig_zag'

Fig. 8: path=[(0, 0), (0.5, 0), (0.5, 0.25), (1, 0.25), (1, 0.75), (0.5, 0.75), (0.5, 1), (0, 1)]

(continued from previous page)

```
                ),
            ]
        ),
        28
    )
]
```

There are two groups in this config: Group 1 is the CurrentLayout widget. Group 2 is the StatusNotifier, Mpris2, Clock and ScriptExit widgets. The groups are separated by the Spacer widget.

When there is no active icon in the StatusNotifier, the bar looks like this:

When there is an icon, the group is expanded to include the widget:

Note the group is not broken despite the Mpris2 widget having no contents.

Groups are determined by looking for:

- widgets using the identical configuration for the decoration
- widgets in a consecutive groups

Groups can therefore be broken by changing the configuration of the group (e.g. by adding an additional parameter such as `group_id=1`) or having an undecorated separator between groups.

Setting `clip=True` will result in the widget's contents being restricted to the area covered by the decoration. This may be desirable for widgets like `ALSAWidget` and `BrightnessControl` which draw their levels in the bar. NB clipping be be constrained to the area inside the outline line width.

Fig. 9: Single decoration

Fig. 10: Two decorations stacked

key	default	description
clip	False	Clip contents of widget to decoration area.
colour	'#000000'	Colour for decoration
extrawidth	0	Add additional width to the end of the decoration
filled	False	Whether to fill shape
group	False	When set to True, the decoration will be applied as if the widgets were grouped. See documentation for more.
ignore_extrawidt	False	Ignores additional width added by decoration. Useful when stacking decorations on top of a PowerLineDecoration.
line_colour	'#ffffff'	Colour of border
line_width	0	Line width for decoration
padding	0	Default padding
padding_x	None	X Padding. Overrides 'padding' if set
padding_y	None	Y Padding. Overrides 'padding' if set
radius	4	Corner radius as int or list of ints [TL TR BR BL]. 0 is square
use_widget_backg	False	Paint the decoration using the colour from the widget's *background* property. The widget's background will then be the bar's background colour.

CHAPTER

TEN

MASKED IMAGES

10.1 ImgMask

class qtile_extras.images.**ImgMask**(**args*, *drawer: Drawer | None = None*, ***kwargs*)

Image object that uses the image source as a mask to paint the background.

Colour can be set at the moment of drawing, rather than preparing images in advance.

10.2 Loader

class qtile_extras.images.**Loader**(**directories*, *masked=True*, ***kwargs*)

Same as libqtile.images.Loader but takes an optional parameter, `masked`, to determine whether to use `ImgMask` class.

CHAPTER

ELEVEN

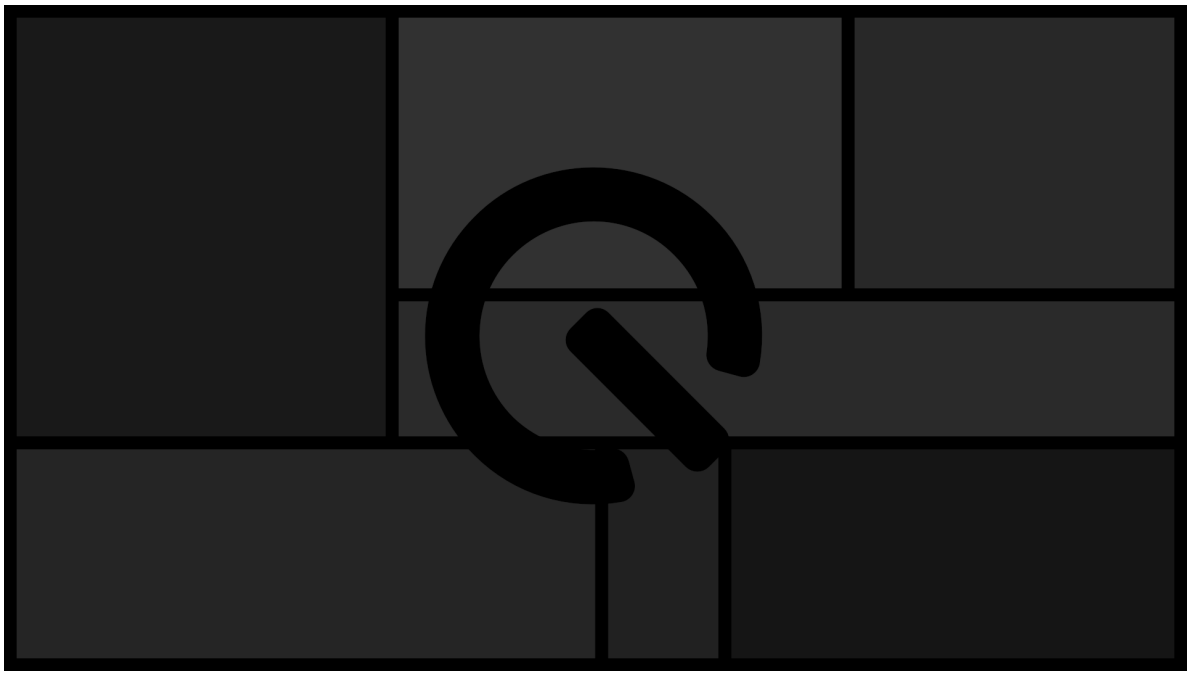
WALLPAPERS

At one point, we thought about shipping Qtile with a default wallpaper as that would be a bit more welcoming than the current black screen. The PR met with mixed reactions so I'll put my "artwork" here instead.

These can be added to your config by doing:

```
from qtile_extras.resources import wallpapers

...

screens = [
    Screen(
        top=Bar(...),
        wallpaper=wallpapers.WALLPAPER_TRIANGLES,
        wallpaper_mode="fill"
    )
]
```

WALLPAPER_TILES



WALLPAPER_TRIANGLES

WALLPAPER_TRIANGLES_ROUNDED

CHAPTER

TWELVE

CHANGELOG

```
2024-01-20: [RELEASE] v0.24.0 - compatible with qtile 0.24.0
2024-01-20: [PYPI] Enable release workflow to push future releases to PyPI
2024-01-20: [BUGFIX] Fix `Visualiser` bug with `hide=True`
2024-01-20: [FEATURE] Add `GroupBox2` widget
2023-12-21: [FEATURE] Add `IWD` widget
2023-11-17: [BUGFIX] Prevent double hooks for Mpris2 widget
2023-11-13: [BUGFIX] A neater fix for the `Visualiser` CPU bug
2023-11-13: [BUGFIX] Fix `fraction` KeyError in `UpowerWidget`
2023-11-13: [FEATURE] Add `invert` option to `Visualiser` to draw bars from top down
2023-11-12: [BUGFIX] (Trying to) fix increasing CPU with `Visualiser` widget
2023-11-11: [DOCS] Update installation instructions
2023-11-10: [FEATURE] Add custom hooks to certain widgets (refer to docs for more)
2023-10-16: [FEATURE] Updated `Bluetooth` widget to add context menu
2023-10-08: [BUGFIX] Fix `BrightnessControl` text display issue in `Bar` mode
2023-10-07: [BUGFIX] Tooltip position on multiple screens
2023-10-07: [BUGFIX] Fix issue with decorations stacked on top of `PowerLineDecoration`␣
↪taking wrong width.
2023-10-01: [BREAKING CHANGE] Update `PulseVolume` and `PulseVolumeExtra` to reflect␣
↪updates pushed to qtile.
                            Needs latest qtile git version.
2023-09-29: [RELEASE] v0.23.0 released - compatible with qtile 0.23.0
2023-09-18: [FEATURE] Add experimental support for `GlobalMenu` in Wayland.
2023-09-17: [TESTS] Check decorations are rendered correctly during test suite
2023-09-16: [FEATURE] Add more popup templates for `Mpris2` widget
2023-09-16: [FEATURE] Add `PulseVolumeExtra` widget
2023-09-01: [BUGFIX] Fix two `RectDecoration` rendering bugs (issues #266 and #267)
2023-08-13: [BUGFIX] Fix `PowerLineDecoration` for Wayland (and make it generally better)
2023-08-10: [BUGFIX] X11 - fix issue with translucent decorations resulting in artefacts␣
↪(needs latest qtile).
2023-08-06: [BREAKING CHANGE] Update imports to align with qtile codebase tidy. You need␣
↪latest qtile git version.
2023-08-04: [BREAKING CHANGE] Fix decorations to work with wlroots0.16 update in qtile.
                            Note: You must be on latest git version of qtile.
2023-04-09: [BUGFIX] Fix menus in `StatusNotifier` for updated electron apps
2023-01-13: [FEATURE] Add support for local icons in `StatusNotifier` context menus
2023-01-03: [FEATURE] Add `PopupCircularProgress` control to popup toolkit
2022-12-31: [BUGFIX] Fix bug in `LiveFootballScores` info popup with excess separators
2022-12-26: [FEATURE] Add decoration grouping to BorderDecorations
2022-12-25: [BUGFIX] Catch errors where `SnapCast` widget can't retrieve ID from server
2022-12-22: [FEATURE] Add `Syncthing` widget
```

(continues on next page)

(continued from previous page)

```
2022-12-21: [BUGFIX] Fix issue redrawing controls after mouse focus change
2022-12-20: [FEATURE] Add `ProgressBarMixin` and convert `ALSAWidget` and␣
↪`BrightnessControl` to use mixin
2022-12-19: [BUGFIX] Update `StravaWidget` to `stravalib v1.1.0` (NB new dependency␣
↪`pint`)
2022-12-03: [FEATURE] Add extended popup to `Mpris2`
2022-12-02: [BUGFUX] Fix startup slowdown caused by `ALSAWidget`
2022-12-02: [BUGHFIX] Fixed bug when drawing masked images with `Image` widget
2022-12-01: [FEATURE] Add ability to fine tune position of `AnalogueClock`
2022-11-29: [BUGFIX] Handle JSON error in `TVHWidget`
2022-11-26: [FEATURE] Add extended popup to `LiveFootballScores`
2022-11-24: [BUGFIX] Fix bug where decoration crashes on widget initialised inside a␣
↪`WidgetBox`
2022-11-23: [FEATURE] Add extended popups to `ALSAWidget` and `BrightnessControl`␣
↪(experimental)
2022-11-22: [FEATURE] Add ability to draw border in different colour in `RectDecoration`
2022-11-22: [BUGFIX] Fix `RectDecoration` grouping when not filled
2022-11-18: [FEATURE] Add support for web-based images in `PopupImage` and `Image` widget
2022-11-18: [FEATURE] Add ability to draw border on `PopupSlider` controls
2022-11-18: [FEATURE] Add `ExtendedPopupMixin` to simplify adding popups to widgets
2022-11-18: [FEATURE] Add `MenuMixin` and `DBusMenuMixin` to standardise approach to␣
↪widget menus
2022-11-17: [FEATURE] Add HTTPDigest authentication to `TVHWidget`
2022-11-13: [FEATURE] Add ability to update popup control values
2022-11-12: [BUGFIX] Fix menu bug in `GlobalMenu`
2022-11-12: [FEATURE] `PopupImage` can display an alternative image as a highlight
2022-11-12: [BUGFIX] Fix away goal and red card indicator locations when using␣
↪`PowerLineDecoration`
2022-11-12: [BUGFIX] More fixes to decorations on mirrored widgets
2022-11-12: [BUGFIX] More `RectDecoration` clipping fixes
2022-11-12: [FEATURE] Add ability to set position of popup relative to corners,␣
↪midpoints and centre of screen
2022-11-11: [DOCS] Update Arch installation instructions
2022-11-11: [BUGFIX] Fix infinite recursion error with decorated widget configuration
2022-11-11: [BUGFIX] Fix `RectDecoration` clipping when widget is resized
2022-11-11: [BUGFIX] Fix menu position for `StatusNotifier` and `GlobalMenu` widgets
2022-11-10: [BUGFIX] Better handling of decorations on mirrored widgets
2022-11-10: [BUGFIX] Fix `Visualiser` segfault
2022-11-08: [FEATURE] Add improvements to `StatusNotifier` menus
2022-11-07: [FEATURE] Add ability to resize icons in `ALSAWidget`
2022-11-06: [BUGFIX] `WiFiIcon` internet check handles other exceptions
2022-11-05: [BUGFIX] Hide `LiveFootballScores` widget when no match
2022-11-01: [BUGFIX] Fix `StatusNotifier` menu bug where submenus do not appear
2022-10-27: [BUGFIX] Fix Graph widgets to work with decorations
2022-10-26: [FEATURE] `WiFiIcon`: Disable background internet check by default
2022-10-25: [BUGFIX] Fix padding in `TooltipMixin`
2022-10-25: [FEATURE] Allow overriding of colours for `PowerLineDecoration`
2022-10-20: [FEATURE] `WiFiIcon`: Add ability to always show SSID
2022-10-20: [FEATURE] `WiFiIcon`: Add test to check whether device is connected to the␣
↪internet
2022-10-16: [FEATURE] Add `ContinuousPoll` widget
2022-10-15: [BUGFIX] Fix `RectDecoration` reload bug with `use_widget_background`
```

(continues on next page)

(continued from previous page)

```
2022-10-10: [UPSTREAM] Change command syntax to match changes to qtile codebase (needs␣
↪latest git version of qtile)
2022-10-05: [BUGFIX] Fix IndexError on grouped `RectDecoration` (issue #126)
2022-10-04: [BUGFIX] Fix `extrawidth` with `PowerlineDecoration`
2022-10-04: [FEATURE] Allow users to add extra width at the end of a decoration.
2022-10-04: [BUGFIX] Fix `Visualizer` image size bug
2022-10-04: [TESTS] Pin `pytest-cov` to 3.0.0 to retain `multiprocessing` support
2022-09-24: [BUGFIX] Improve icon background rendering in `Systray` with `RectDecoration`
2022-02-22: [RELEASE] v0.22.1 (skipped v0.22.0)
2022-09-22: [BUGFIX] Stop multiple visualiser processes being spawned by widget
2022-09-19: [FEATURE] Add `Visualizer` widget
2022-09-16: [BUGFIX] Fix transparency issue in `PowerLineDecoration`
2022-09-13: [FEATURE] Add clipping to `RectDecoration`
2022-09-11: [FEATURE] Add grouping to `RectDecoration`
2022-09-11: [BUGFIX] Fix mouse callback bug in `WifiIcon` widget
2022-08-24: [BUGFIX] Better handling of popup finalisation
2022-08-19: [BUGFIX] Unsubscribe signal callbacks in `Upower` widget on restart
2022-08-19: [BUGFIX] Fix bug where `GlobalMenu` is not shown for current window after␣
↪restart
2022-08-19: [BUGFIX] Fix menu position for `StatusNotifier` for multiple monitors
2022-08-18: [FEATURE] Add `PowerLineDecoration`
2022-07-27: [FEATURE] Add `fill_charge` to UPower widget
2022-07-21: [BUGFIX] Fix CurrentLayoutIcon to work with latest git version
2022-07-21: [BUGFIX] Fix blocking API calls in Snapcast and GithubNotification widgets
2022-07-07: [BUGFIX] Fix UPowerWidget not showing text on click when fontsize is None
2022-07-03: [PACKAGING] Use git versioning details
2022-07-03: [BUGFIX] Fix scaling on CurrentLayoutIcon and disable `use_mask=True` default
2022-07-02: [PACKAGING] Fix missing files in package
2022-07-02: [BUGFIX] Fix WordClock crash at certain times
2022-07-01: [FEATURE] Add CurrentLayoutIcon widget (with colour icons)
2022-06-28: [FEATURE] Add GithubNotifications widget
2022-06-28: [BUGFIX] Fix keyboard navigation for toolkit (must be on latest git version␣
↪of qtile)
2022-05-26: [CONFIG_BREAK] Use `foreground` to set font colour, rather than `font_colour`
2022-05-08: [FEATURE] Add GlobalMenu widget (alpha version!)
2022-05-08: [FEATURE] Add AnalogueClock widget
2022-04-29: [FEATURE] Add ability to display widgets in popup window
2022-03-16: [BUGFIX] Better positioning of popups on multiscreen setups (#50)
2022-03-15: [BUGFIX] Ensure toolkit disables naviagation when no focusable controls (#49)
2022-03-15: [DOCS] Fix docs build error (git --> https)
2022-03-05: [GITHUB] Require PRs to update CHANGELOG
2022-03-05: [BUGFIX] Fix tooltip only showing once
2022-03-05: [BUGFIX] Better handling of empty menus in StatusNotifier widget
2022-01-09: [FEATURE] Add SnapCast widget (missing some functionality)
```

CHAPTER

THIRTEEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

V

W