qtawesome Documentation

Release 1.2.3

The Spyder Development Team

Mar 09, 2023

CONTENTS

CHAPTER

ONE

INTRODUCTION

QtAwesome enables iconic fonts such as Font Awesome and Elusive Icons in PyQt and PySide applications.

It started as a Python port of the QtAwesome C++ library by Rick Blommers.

CHAPTER

TWO

USAGE

2.1 Supported Fonts

QtAwesome identifies icons by their **prefix** and their **icon name**, separated by a *period* (`.`) character.

The following prefixes are currently available to use:

- FontAwesome:
  - FA 5.15.4 features 1,608 free icons in different styles:
    - `fa5` prefix has 151 icons in the "regular" style.
    - `fa5s` prefix has 1001 icons in the "solid" style.
    - `fa5b` prefix has 456 icons of various brands.
  - `fa` is the legacy FA 4.7 version with its 675 icons but **all** of them (*and more!*) are part of FA 5.x so you should probably use the newer version above.
- `ei` prefix holds Elusive Icons 2.0 with its 304 icons.
- Material Design Icons:
  - `mdi6` prefix holds Material Design Icons 6.3.95 with its 6395 icons.
  - `mdi` prefix holds Material Design Icons 5.9.55 with its 5955 icons.
- `ph` prefix holds Phosphor 1.3.0 with its 4470 icons (894 icons * 5 weights: Thin, Light, Regular, Bold and Fill).
- `ri` prefix holds Remix Icon 2.5.0 with its 2271 icons.
- `msc` prefix holds Microsoft's Codicons 0.0.32 with its 421 icons.

2.2 Examples

```
import qtawesome as qta
```

- Use Font Awesome, Elusive Icons, Material Design Icons, Phosphor, Remix Icon or Microsoft's Codicons:

```
# Get FontAwesome 5.x icons by name in various styles by name
fa5_icon = qta.icon('fa5.flag')
fa5_button = QtWidgets.QPushButton(fa5_icon, 'Font Awesome! (regular)')

fa5s_icon = qta.icon('fa5s.flag')
fa5s_button = QtWidgets.QPushButton(fa5s_icon, 'Font Awesome! (solid)')
```

(continues on next page)

(continued from previous page)

```
fa5b_icon = qta.icon('fa5b.github')
fa5b_button = QtWidgets.QPushButton(fa5b_icon, 'Font Awesome! (brands)')

# Get Elusive icons by name
asl_icon = qta.icon('ei.asl')
elusive_button = QtWidgets.QPushButton(asl_icon, 'Elusive Icons!')

# Get Material Design icons 6.x by name
apn_icon = qta.icon('mdi6.access-point-network')
mdi6_button = QtWidgets.QPushButton(apn_icon, 'Material Design Icons!')

# Get Phosphor by name
mic_icon = qta.icon('ph.microphone-fill')
ph_button = QtWidgets.QPushButton(mic_icon, 'Phosphor!')

# Get Remix Icon by name
truck_icon = qta.icon('ri.truck-fill')
ri_button = QtWidgets.QPushButton(truck_icon, 'Remix Icon!')

# Get Microsoft's Codicons by name
squirrel_icon = qta.icon('msc.squirrel')
msc_button = QtWidgets.QPushButton(squirrel_icon, 'Codicons!')
```

- Apply some transformations:

```
# Rotated
rot_icon = qta.icon('mdi.access-point-network', rotated=45)
rot_button = QtWidgets.QPushButton(rot_icon, 'Rotated Icons!')

# Horizontal flip
hflip_icon = qta.icon('mdi.account-alert', hflip=True)
hflip_button = QtWidgets.QPushButton(hflip_icon, 'Horizontally Flipped Icons!')

# Vertical flip
vflip_icon = qta.icon('mdi.account-alert', vflip=True)
vflip_button = QtWidgets.QPushButton(vflip_icon, 'Vertically Flipped Icons!')
```

- Apply some styling:

```
# Styling
styling_icon = qta.icon('fa5s.music',
                        active='fa5s.balance-scale',
                        color='blue',
                        color_active='orange')
music_button = QtWidgets.QPushButton(styling_icon, 'Styling')
```

- Set alpha in colors:

```
# Setting an alpha of 120 to the color of this icon. Alpha must be a number
# between 0 and 255.
icon_with_alpha = qta.icon('mdi.heart',
```

(continues on next page)

(continued from previous page)

```
                            color=('red', 120))
heart_button = QtWidgets.QPushButton(icon_with_alpha, 'Setting alpha')
```

- Apply toggling state styling:

```
# Toggle
toggle_icon = qta.icon('fa5s.home', selected='fa5s.balance-scale',
                       color_off='black',
                       color_off_active='blue',
                       color_on='orange',
                       color_on_active='yellow')
toggle_button = QtWidgets.QPushButton(toggle_icon, 'Toggle')
toggle_button.setCheckable(True)
```

- Define the way to draw icons (*text*- default for icons without animation, *path* - default for icons with animations, *glyphrun* and *image*):

```
# Icon drawn with the `image` option
drawn_image_icon = qta.icon('ri.truck-fill',
                            options=[{'draw': 'image'}])
drawn_image_button = QtWidgets.QPushButton(drawn_image_icon,
                                           'Icon drawn as an image')
```

- Stack multiple icons:

```
# Stack icons
camera_ban = qta.icon('fa5s.camera', 'fa5s.ban',
                      options=[{'scale_factor': 0.5,
                                'active': 'fa5s.balance-scale'},
                               {'color': 'red', 'opacity': 0.7}])
stack_button = QtWidgets.QPushButton(camera_ban, 'Stack')
stack_button.setIconSize(QtCore.QSize(32, 32))

# Stack and offset icons
saveall = qta.icon('fa5.save', 'fa5.save',
                   options=[{'scale_factor': 0.8,
                             'offset': (0.2, 0.2),
                             'color': 'gray'},
                            {'scale_factor': 0.8}])
saveall_button = QtWidgets.QPushButton(saveall, 'Stack, offset')
```

- Animations:

```
# Spin icons
spin_button = QtWidgets.QPushButton(' Spinning icon')
spin_icon = qta.icon('fa5s.spinner', color='red',
                     animation=qta.Spin(spin_button))
spin_button.setIcon(spin_icon)

# Pulse icons
pulse_button = QtWidgets.QPushButton(' Pulsing icon')
pulse_icon = qta.icon('fa5s.spinner', color='green',
                      animation=qta.Pulse(pulse_button))
```

(continues on next page)

(continued from previous page)

```
pulse_button.setIcon(pulse_icon)

# Stacked spin icons
stack_spin_button = QtWidgets.QPushButton('Stack spin')
options = [{'scale_factor': 0.4,
            'animation': qta.Spin(stack_spin_button)},
           {'color': 'blue'}]
stack_spin_icon = qta.icon('ei.asl', 'fa5.square',
                           options=options)
stack_spin_button.setIcon(stack_spin_icon)
stack_spin_button.setIconSize(QtCore.QSize(32, 32))
```

- Apply font label rendering:

```
# Render a label with this font
label = QtWidgets.QLabel(unichr(0xf19c) + ' ' + 'Label')
label.setFont(qta.font('fa', 16))
```

- Display Icon as a widget:

```
# Spining icon widget
spin_widget = qta.IconWidget()
spin_icon = qta.icon('mdi.loading', color='red',
                     animation=qta.Spin(spin_widget))
spin_widget.setIcon(spin_icon)

# simple widget
simple_widget = qta.IconWidget('mdi.web', color='blue')
```

2.3 Screenshot

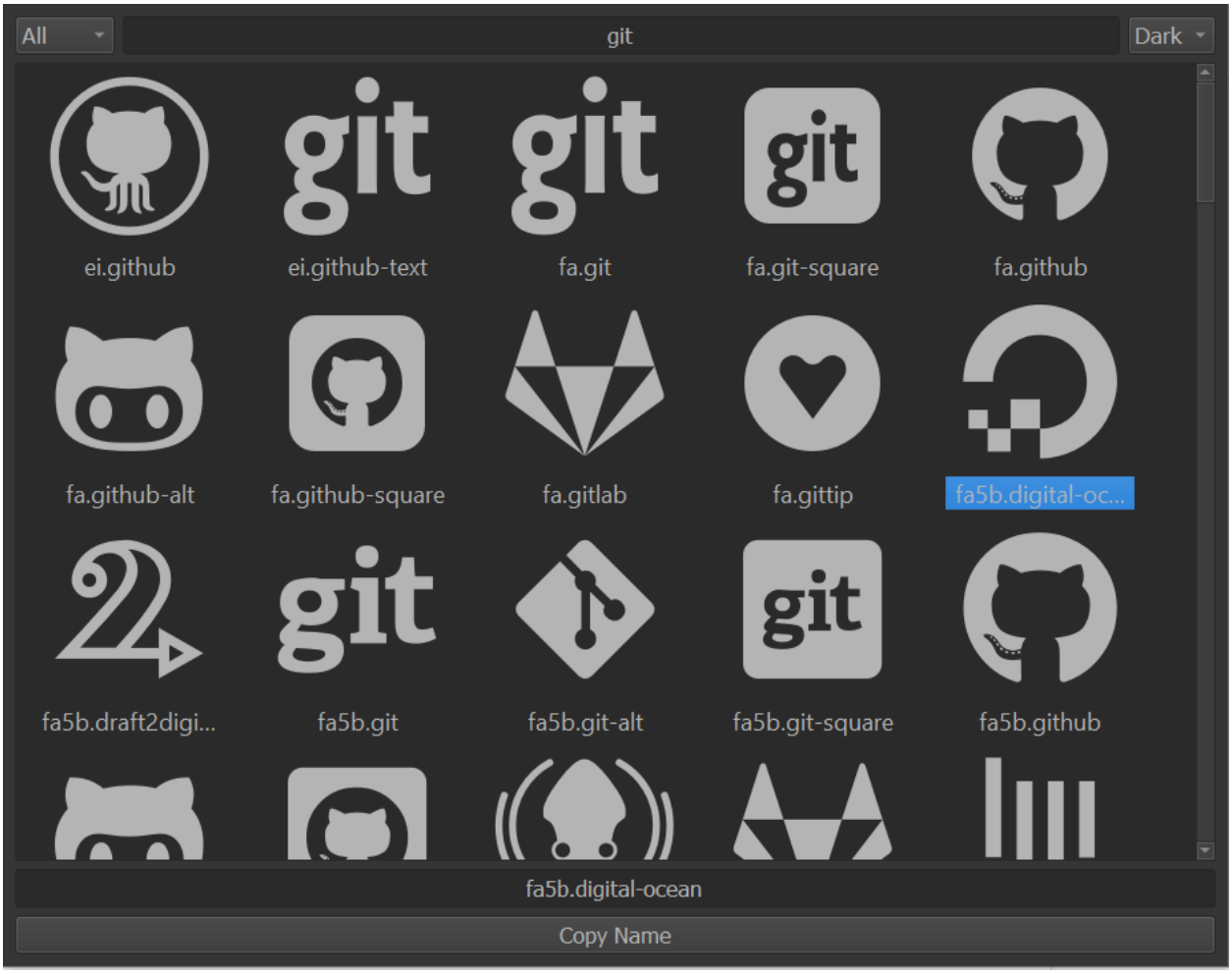
CHAPTER

THREE

ICON BROWSER

QtAwesome ships with a browser that displays all the available icons. You can use this to search for an icon that suits your requirements and then copy the name that should be used to create that icon!

Once installed, run *qta-browser* from a shell to start the browser.



CHAPTER

FOUR

API REFERENCE DOCUMENTATION

4.1 qtawesome

Font-Awesome and other iconic fonts for PyQt / PySide applications.

icon(*names, **kwargs)	Return a QIcon object corresponding to the provided icon name(s).
load_font(prefix, ttf_filename, charmap_filename)	Loads a font file and the associated charmap.
charmap(prefixed_name)	Return the character map used for a given font.
font(prefix, size)	Return the font corresponding to the specified prefix.
set_defaults(**kwargs)	Set default options for icons.

4.1.1 qtawesome.icon

qtawesome.**icon**(**names*, ***kwargs*)

Return a QIcon object corresponding to the provided icon name(s).

This function is the main interface of qtawesome.

It can be used to create a QIcon instance from a single glyph or from a list of glyphs that are displayed on the top of each other. Such icon stacks are generally used to combine multiple glyphs to make more complex icons.

Glyph names are specified by strings, of the form `prefix.name`. The `prefix` corresponds to the font to be used and `name` is the name of the icon.

- The prefix corresponding to Font-Awesome 4.x is 'fa'
- The prefix corresponding to Font-Awesome 5.x (regular) is 'fa5'
- The prefix corresponding to Font-Awesome 5.x (solid) is 'fa5s'
- The prefix corresponding to Font-Awesome 5.x (brands) is 'fa5b'
- The prefix corresponding to Elusive-Icons is 'ei'
- The prefix corresponding to Material-Design-Icons 5.x is 'mdi'
- The prefix corresponding to Material-Design-Icons 6.x is 'mdi6'
- The prefix corresponding to Phosphor is 'ph'
- The prefix corresponding to Remix-Icon is 'ri'
- The prefix corresponding to Microsoft's Codicons is 'msc'

When requesting a single glyph, options (such as color or positional offsets) can be passed as keyword arguments:

```
import qtawesome as qta

music_icon = qta.icon(
    'fa5s.music',
    color='blue',
    color_active='orange')
```

When requesting multiple glyphs, the *options* keyword argument contains the list of option dictionaries to be used for each glyph:

```
camera_ban = qta.icon('fa5s.camera', 'fa5s.ban', options=[{
        'scale_factor': 0.5,
        'active': 'fa5s.balance-scale'
    }, {
        'color': 'red',
        'opacity': 0.7
    }])
```

Qt's `QIcon` object has four modes

- `Normal`: The user is not interacting with the icon, but the functionality represented by the icon is available.
- `Disabled`: The functionality corresponding to the icon is not available.
- `Active`: The functionality corresponding to the icon is available. The user is interacting with the icon, for example, moving the mouse over it or clicking it.
- `Selected`: The item represented by the icon is selected.

The glyph for the Normal mode is the one specified with the main positional argument.

- `color`: icon color in the `Normal` mode.

The following four options will apply to the icon regardless of the mode.

- `offset`: tuple (x, y) corresponding to the horizontal and vertical offsets for the glyph, specified as a proportion of the icon size.
- `animation`: animation object for the icon.
- `scale_factor`: multiplicative scale factor to be used for the glyph.

The following options apply to the different modes of the icon

- `active`: name of the glyph to be used when the icon is `Active`.
- `color_active`: the corresponding icon color.
- `disabled`: name of the glyph to be used when the icon is `Disabled`.
- `color_disabled`: the corresponding icon color.
- `selected`: name of the glyph to be used when the icon is `Selected`.
- `color_selected`: the corresponding icon color.

Default values for these options can be specified via the function `set_defaults`. If unspecified, the default defaults are:

```
{
    'opacity': 1.0,
    'scale_factor': 1.0
```

(continues on next page)

(continued from previous page)

```
    'color': QColor(50, 50, 50),
    'color_disabled': QColor(150, 150, 150),
}
```

If no default value is provided for `active`, `disabled` or `selected` the glyph specified for the `Normal` mode will be used.

4.1.2 qtawesome.load_font

qtawesome.**load_font**(*prefix*, *ttf_filename*, *charmap_filename*, *directory=None*)

Loads a font file and the associated charmap.

If `directory` is passed, the files will be looked for in the qtawesome `fonts` directory.

Parameters

- **prefix** (*str*) – Prefix string to be used when accessing a given font set
- **ttf_filename** (*str*) – Ttf font filename
- **charmap_filename** (*str*) – Character map filename
- **directory** (*str or None, optional*) – Directory path for font and charmap files

Example

If you want to load a font `myicon.tff` with a `myicon-charmap.json` charmap added to the qtawesome `fonts` directory (usually located at `</path/to/lib/python>/site-packages/qtawesome/fonts/`) you can use:

```
qta.load_font(
    'myicon',
    'myicon.ttf',
    'myicon-charmap.json'
)
```

However, if you want to load a font `myicon.tff` with a `myicon-charmap.json` charmap located in a specific path outside the qtawesome `font` directory like for example `/path/to/myproject/fonts` you can use:

```
qta.load_font(
    'myicon',
    'myicon.ttf',
    'myicon-charmap.json',
    directory='/path/to/myproject/fonts'
)
```

4.1.3 qtawesome.charmap

qtawesome.**charmap**(*prefixed_name*)

Return the character map used for a given font.

Returns
return_value – The dictionary mapping the icon names to the corresponding unicode character.

Return type
dict

4.1.4 qtawesome.font

qtawesome.**font**(*prefix*, *size*)

Return the font corresponding to the specified prefix.

This can be used to render text using the iconic font directly:

```
import qtawesome as qta
from qtpy import QtWidgets

label = QtWidgets.QLabel(unichr(0xf19c) + ' ' + 'Label')
label.setFont(qta.font('fa', 16))
```

Parameters

- **prefix** (*str*) – prefix string of the loaded font
- **size** (*int*) – size for the font

4.1.5 qtawesome.set_defaults

qtawesome.**set_defaults**(***kwargs*)

Set default options for icons.

The valid keyword arguments are:

'active', 'animation', 'color', 'color_active', 'color_disabled', 'color_selected', 'disabled', 'offset', 'scale_factor', 'selected'.

PYTHON MODULE INDEX

INDEX