qutebrowser extensions Documentation

Florian Bruhin

Dec 12, 2018

CONTENTS:

CHAPTER

ONE

API MODULES

1.1 cmdutils module

qutebrowser has the concept of functions which are exposed to the user as commands.

Creating a new command is straightforward:

```
from qutebrowser.api import cmdutils

@cmdutils.register(...)
def foo():
    ...
```

The commands arguments are automatically deduced by inspecting your function.

The types of the function arguments are inferred based on their default values, e.g., an argument *foo=True* will be converted to a flag *-f/–foo* in qutebrowser's commandline.

The type can be overridden using Python's function annotations:

```
@cmdutils.register(...)
def foo(bar: int, baz=True):
    ...
```

Possible values:

- A callable (`int`, `float`, etc.): Gets called to validate/convert the value.
- A python enum type: All members of the enum are possible values.
- A `typing.Union` of multiple types above: Any of these types are valid values, e.g., `typing.Union[str, int]`.

exception qutebrowser.api.cmdutils.**CommandError**

Raised when a command encounters an error while running.

If your command handler encounters an error and cannot continue, raise this exception with an appropriate error message:

```
raise cmdexc.CommandError("Message")
```

The message will then be shown in the qutebrowser status bar.

Note: You should only raise this exception while a command handler is run. Raising it at another point causes qutebrowser to crash due to an unhandled exception.

---

qutebrowser.api.cmdutils.**check_overflow**(*arg: int*, *ctype: str*) → None
Check if the given argument is in bounds for the given type.

Parameters

- **arg** – The argument to check.
- **ctype** – The C++/Qt type to check as a string ('int'/'int64').

qutebrowser.api.cmdutils.**check_exclusive**(*flags: Iterable[bool]*, *names: Iterable[str]*) → None
Check if only one flag is set with exclusive flags.

Raise a CommandError if not.

Parameters

- **flags** – The flag values to check.
- **names** – A list of names (corresponding to the flags argument).

class qutebrowser.api.cmdutils.**register**(*, *instance: str = None*, *name: str = None*, ***kwargs*)
Decorator to register a new command handler.

class qutebrowser.api.cmdutils.**argument**(*argname: str*, ***kwargs*)
Decorator to customize an argument.

You can customize how an argument is handled using the `@cmdutils.argument` decorator *after* `@cmdutils.register`. This can, for example, be used to customize the flag an argument should get:

```
@cmdutils.register(...)
@cmdutils.argument('bar', flag='c')
def foo(bar):
    ...
```

For a `str` argument, you can restrict the allowed strings using `choices`:

```
@cmdutils.register(...)
@cmdutils.argument('bar', choices=['val1', 'val2'])
def foo(bar: str):
    ...
```

For `typing.Union` types, the given `choices` are only checked if other types (like `int`) don't match.

The following arguments are supported for `@cmdutils.argument`:

- `flag`: Customize the short flag (`-x`) the argument will get.
- `value`: Tell qutebrowser to fill the argument with special values:
  - `value=cmdutils.Value.count`: The `count` given by the user to the command.
  - `value=cmdutils.Value.win_id`: The window ID of the current window.
  - `value=cmdutils.Value.cur_tab`: The tab object which is currently focused.
- `completion`: A completion function to use when completing arguments for the given command.
- `choices`: The allowed string choices for the argument.

The name of an argument will always be the parameter name, with any trailing underscores stripped and underscores replaced by dashes.

class qutebrowser.api.cmdutils.**KeyMode**
Key input modes.

normal = 1
Normal mode (no mode was entered)

hint = 2
Hint mode (showing labels for links)

command = 3
Command mode (after pressing the colon key)

yesno = 4
Yes/No prompts

prompt = 5
Text prompts

insert = 6
Insert mode (passing through most keys)

passthrough = 7
Passthrough mode (passing through all keys)

caret = 8
Caret mode (moving cursor with keys)

qutebrowser.api.cmdutils.**Value**
alias of qutebrowser.utils.usertypes.CommandValue

1.2 apitypes module

A single tab.

class qutebrowser.api.apitypes.**ClickTarget**
How to open a clicked link.

normal = 0
Open the link in the current tab

tab = 1
Open the link in a new foreground tab

tab_bg = 2
Open the link in a new background tab

window = 3
Open the link in a new window

hover = 4
Only hover over the link

class qutebrowser.api.apitypes.**InitContext**(*data_dir*, *config_dir*, *args*)
Context an extension gets in its init hook.

class qutebrowser.api.apitypes.**JsWorld**
World/context to run JavaScript code in.

main = 1
Same world as the web page's JavaScript.

application = 2
Application world, used by qutebrowser internally.

user = 3
User world, currently not used.

jseval = 4
World used for the jseval-command.

qutebrowser.api.apitypes.**Tab**
alias of *qutebrowser.browser.browsertab.AbstractTab*

qutebrowser.api.apitypes.**WebElemError**
alias of *qutebrowser.browser.webelem.Error*

qutebrowser.api.apitypes.**WebElement**
alias of *qutebrowser.browser.webelem.AbstractWebElement*

exception qutebrowser.api.apitypes.**WebTabError**
Base class for various errors.

1.3 config module

Access to the qutebrowser configuration.

qutebrowser.api.config.**val = None**
Simplified access to config values using attribute acccess. For example, to access the `content.javascript.enabled` setting, you can do:

```
if config.val.content.javascript.enabled:
    ...
```

This also supports setting configuration values:

```
config.val.content.javascript.enabled = False
```

qutebrowser.api.config.**get**(*name: str*, *url: PyQt5.QtCore.QUrl = None*) → Any
Get a value from the config based on a string name.

1.4 downloads module

APIs related to downloading files.

class qutebrowser.api.downloads.**TempDownload**(*item: qutebrowser.browser.qtnetworkdownloads.DownloadItem*)
A download of some data into a file object.

qutebrowser.api.downloads.**download_temp**(*url: PyQt5.QtCore.QUrl*) → qutebrowser.api.downloads.TempDownload
Download the given URL into a file object.

The download is not saved to disk.

Returns a `TempDownload` object, which triggers a `finished` signal when the download has finished:

```
dl = downloads.download_temp(QUrl("https://www.example.com/"))
dl.finished.connect(functools.partial(on_download_finished, dl))
```

After the download has finished, its `successful` attribute can be checked to make sure it finished successfully. If so, its contents can be read by accessing the `fileobj` attribute:

```
def on_download_finished(download: downloads.TempDownload) -> None:
    if download.successful:
        print(download.fileobj.read())
        download.fileobj.close()
```

1.5 hook module

Hooks for extensions.

class qutebrowser.api.hook.**init**
Decorator to mark a function to run when initializing.

The decorated function gets called with a *qutebrowser.api.apitypes.InitContext* as argument.

Example:

```
@hook.init()
def init(_context):
    message.info("Extension initialized.")
```

class qutebrowser.api.hook.**config_changed**(*option_filter: str = None*)
Decorator to get notified about changed configs.

By default, the decorated function is called when any change in the config occurs:

```
@hook.config_changed()
def on_config_changed():
    ...
```

When an option name is passed, it's only called when the given option was changed:

```
@hook.config_changed('content.javascript.enabled')
def on_config_changed():
    ...
```

Alternatively, a part of an option name can be specified. In the following snippet, `on_config_changed` gets called when either `bindings.commands` or `bindings.key_mappings` have changed:

```
@hook.config_changed('bindings')
def on_config_changed():
    ...
```

1.6 interceptor module

APIs related to intercepting/blocking requests.

qutebrowser.api.interceptor.**register**(*interceptor: Callable[[qutebrowser.extensions.interceptors.Request], None]*) → None
Register a request interceptor.

Whenever a request happens, the interceptor gets called with a *Request* object.

Example:

```
def intercept(request: interceptor.Request) -> None:
    if request.request_url.host() == 'badhost.example.com':
        request.block()
```

class qutebrowser.api.interceptor.**Request**(*first_party_url*, *request_url*, *is_blocked=False*)
A request which can be intercepted/blocked.

first_party_url = None
The URL of the page being shown.

request_url = None
The URL of the file being requested.

block() → None
Block this request.

1.7 message module

Utilities to display messages above the status bar.

qutebrowser.api.message.**error**(*message: str*, *, *stack: str = None*, *replace: bool = False*) → None
Display an error message.

Parameters

- **message** – The message to show.
- **stack** – The stack trace to show (if any).
- **replace** – Replace existing messages which are still being shown.

qutebrowser.api.message.**info**(*message: str*, *, *replace: bool = False*) → None
Display an info message.

Parameters

- **message** – The message to show.
- **replace** – Replace existing messages which are still being shown.

qutebrowser.api.message.**warning**(*message: str*, *, *replace: bool = False*) → None
Display a warning message.

Parameters

- **message** – The message to show.
- **replace** – Replace existing messages which are still being shown.

CHAPTER

TWO

TAB API

class qutebrowser.browser.browsertab.**AbstractTab**

An adapter for QWebView/QWebEngineView representing a single tab.

window_close_requested
Signal emitted when a website requests to close this tab.

link_hovered
Signal emitted when a link is hovered (the hover text)

load_started
Signal emitted when a page started loading

load_progress
Signal emitted when a page is loading (progress percentage)

load_finished
Signal emitted when a page finished loading (success as bool)

icon_changed
Signal emitted when a page's favicon changed (icon as QIcon)

title_changed
Signal emitted when a page's title changed (new title as str)

new_tab_requested
Signal emitted when a new tab should be opened (url as QUrl)

url_changed
Signal emitted when a page's URL changed (url as QUrl)

contents_size_changed
Signal emitted when a tab's content size changed (new size as QSizeF)

fullscreen_requested
Signal emitted when a page requested full-screen (bool)

before_load_started
Signal emitted before load starts (URL as QUrl)

send_event(*evt: PyQt5.QtCore.QEvent*) → None
Send the given event to the underlying widget.

The event will be sent via QApplication.postEvent. Note that a posted event must not be re-used in any way!

fake_key_press(*key: PyQt5.QtCore.Qt.Key*, *modifier: PyQt5.QtCore.Qt.KeyboardModifier = 0*) → None
Send a fake key event to this tab.

dump_async(*callback: Callable[[str], None], *, plain: bool = False*) → None
Dump the current page's html asynchronously.

The given callback will be called with the result when dumping is complete.

run_js_async(*code: str, callback: Callable[[Any], None] = None, *, world: Union[qutebrowser.utils.usertypes.JsWorld, int] = None*) → None
Run javascript async.

The given callback will be called with the result when running JS is complete.

Parameters

- **code** – The javascript code to run.
- **callback** – The callback to call with the result, or None.
- **world** – A world ID (int or usertypes.JsWorld member) to run the JS in the main world or in another isolated world.

class qutebrowser.browser.browsertab.**AbstractAction**
Attribute `action` of AbstractTab for Qt WebActions.

exit_fullscreen() → None
Exit the fullscreen mode.

save_page() → None
Save the current page.

run_string(*name: str*) → None
Run a webaction based on its name.

show_source(*pygments: bool = False*) → None
Show the source of the current page in a new tab.

class qutebrowser.browser.browsertab.**AbstractPrinting**
Attribute `printing` of AbstractTab for printing the page.

check_pdf_support() → None
Check whether writing to PDFs is supported.

If it's not supported (by the current Qt version), a WebTabError is raised.

check_printer_support() → None
Check whether writing to a printer is supported.

If it's not supported (by the current Qt version), a WebTabError is raised.

check_preview_support() → None
Check whether showing a print preview is supported.

If it's not supported (by the current Qt version), a WebTabError is raised.

to_pdf(*filename: str*) → bool
Print the tab to a PDF with the given filename.

to_printer(*printer: PyQt5.QtPrintSupport.QPrinter, callback: Callable[[bool], None] = None*) → None
Print the tab.

Parameters

- **printer** – The QPrinter to print to.
- **callback** – Called with a boolean (True if printing succeeded, False otherwise)

show_dialog() → None
Print with a QPrintDialog.

class qutebrowser.browser.browsertab.**AbstractSearch**
Attribute search of AbstractTab for doing searches.

text
The last thing this view was searched for.

search_displayed
Whether we're currently displaying search results in this view.

_flags
The flags of the last search (needs to be set by subclasses).

_widget
The underlying WebView widget.

finished
Signal emitted when a search was finished (True if the text was found, False otherwise)

cleared
Signal emitted when an existing search was cleared.

search(*text: str, *, ignore_case: qutebrowser.utils.usertypes.IgnoreCase = <IgnoreCase.never: 2>, reverse: bool = False, result_cb: Callable[[bool], None] = None*) → None
Find the given text on the page.

Parameters

- **text** – The text to search for.
- **ignore_case** – Search case-insensitively.
- **reverse** – Reverse search direction.
- **result_cb** – Called with a bool indicating whether a match was found.

clear() → None
Clear the current search.

prev_result(*, *result_cb: Callable[[bool], None] = None*) → None
Go to the previous result of the current search.

Parameters **result_cb** – Called with a bool indicating whether a match was found.

next_result(*, *result_cb: Callable[[bool], None] = None*) → None
Go to the next result of the current search.

Parameters **result_cb** – Called with a bool indicating whether a match was found.

class qutebrowser.browser.browsertab.**AbstractZoom**
Attribute zoom of AbstractTab for controlling zoom.

apply_offset(*offset: int*) → None
Increase/Decrease the zoom level by the given offset.

Parameters **offset** – The offset in the zoom level list.

Returns The new zoom percentage.

set_factor(*factor: float*, *, *fuzzyval: bool = True*) → None
Zoom to a given zoom factor.

Parameters

- **factor** – The zoom factor as float.

- **fuzzyval** – Whether to set the NeighborLists fuzzyval.

class qutebrowser.browser.browsertab.**AbstractCaret**
Attribute `caret` of AbstractTab for caret browsing.

selection_toggled
Signal emitted when the selection was toggled. (argument - whether the selection is now active)

follow_selected_done
Emitted when a `follow_selection` action is done.

class qutebrowser.browser.browsertab.**AbstractScroller**
Attribute `scroller` of AbstractTab to manage scroll position.

perc_changed
Signal emitted when the scroll position changed (int, int)

before_jump_requested
Signal emitted before the user requested a jump. Used to set the special ‘ mark so the user can return.

class qutebrowser.browser.browsertab.**AbstractHistory**
The history attribute of a AbstractTab.

back(*count: int = 1*) → None
Go back in the tab's history.

forward(*count: int = 1*) → None
Go forward in the tab's history.

class qutebrowser.browser.browsertab.**AbstractElements**
Finding and handling of elements on the page.

find_css(*selector: str, callback: Callable[[Sequence[webelem.AbstractWebElement]], None], error_cb: Callable[[Exception], None], *, only_visible: bool = False*) → None
Find all HTML elements matching a given selector async.

If there's an error, the callback is called with a webelem.Error instance.

Parameters

- **callback** – The callback to be called when the search finished.
- **error_cb** – The callback to be called when an error occurred.
- **selector** – The CSS selector to search for.
- **only_visible** – Only show elements which are visible on screen.

find_id(*elem_id: str, callback: Callable[[Optional[webelem.AbstractWebElement]], None]*) → None
Find the HTML element with the given ID async.

Parameters

- **callback** – The callback to be called when the search finished. Called with a WebEngineElement or None.
- **elem_id** – The ID to search for.

find_focused(*callback: Callable[[Optional[webelem.AbstractWebElement]], None]*) → None
Find the focused element on the page async.

Parameters **callback** – The callback to be called when the search finished. Called with a WebEngineElement or None.

find_at_pos(*pos: PyQt5.QtCore.QPoint, callback: Callable[[Optional[webelem.AbstractWebElement]], None]*) → None

Find the element at the given position async.

This is also called "hit test" elsewhere.

Parameters

- **pos** – The QPoint to get the element for.
- **callback** – The callback to be called when the search finished. Called with a WebEngineElement or None.

class qutebrowser.browser.browsertab.**AbstractAudio**

Handling of audio/muting for this tab.

set_muted(*muted: bool*, *override: bool* = *False*) → None

Set this tab as muted or not.

Parameters **override** – If set to True, muting/unmuting was done manually and overrides future automatic mute/unmute changes based on the URL.

is_recently_audible() → bool

Whether this tab has had audio playing recently.

CHAPTER

THREE

WEB ELEMENT API

class qutebrowser.browser.webelem.**AbstractWebElement**(*tab*)

A wrapper around QtWebKit/QtWebEngine web element.

tab

The tab associated with this element.

has_frame()

Check if this element has a valid frame attached.

geometry()

Get the geometry for this element.

classes()

Get a list of classes assigned to this element.

tag_name()

Get the tag name of this element.

The returned name will always be lower-case.

outer_xml()

Get the full HTML representation of this element.

value()

Get the value attribute for this element, or None.

set_value(*value*)

Set the element value.

dispatch_event(*event*, *bubbles=False*, *cancelable=False*, *composed=False*)

Dispatch an event to the element.

Parameters

- **bubbles** – Whether this event should bubble.
- **cancelable** – Whether this event can be cancelled.
- **composed** – Whether the event will trigger listeners outside of a shadow root.

insert_text(*text*)

Insert the given text into the element.

rect_on_view(*, *elem_geometry=None*, *no_js=False*)

Get the geometry of the element relative to the webview.

Parameters

- **elem_geometry** – The geometry of the element, or None.

- **no_js** – Fall back to the Python implementation.

is_writable()
Check whether an element is writable.

is_content_editable()
Check if an element has a contenteditable attribute.

Parameters **elem** – The QWebElement to check.

Returns True if the element has a contenteditable attribute, False otherwise.

is_editable(*strict=False*)
Check whether we should switch to insert mode for this element.

Parameters **strict** – Whether to do stricter checking so only fields where we can get the value match, for use with the :editor command.

Returns True if we should switch to insert mode, False otherwise.

is_text_input()
Check if this element is some kind of text box.

remove_blank_target()
Remove target from link.

resolve_url(*baseurl*)
Resolve the URL in the element's src/href attribute.

Parameters **baseurl** – The URL to base relative URLs on as QUrl.

Returns A QUrl with the absolute URL, or None.

is_link()
Return True if this AbstractWebElement is a link.

click(*click_target*, *, *force_event=False*)
Simulate a click on the element.

Parameters

- **click_target** – A usertypes.ClickTarget member, what kind of click to simulate.
- **force_event** – Force generating a fake mouse event.

hover()
Simulate a mouse hover over the element.

class qutebrowser.browser.webelem.**Error**
Base class for WebElement errors.

class qutebrowser.browser.webelem.**OrphanedError**
Raised when a webelement's parent has vanished.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

q

INDEX

Symbols

A

B

C

D