app.bundle() -> str: Get active app's bundle identifier

app.executable() -> str: Get active app's executable name

app.name() -> str: Get active app's name

app.notify(body: str = '', title: str = '', subtitle: str = '', sound: bool = False): Show a desktop notification

app.path() -> str: Get active app's file path

app.preferences(): Open app preferences

app.tab\_close(): Close the current tab

app.tab\_detach(): Move the current tab to a new window

app.tab\_next(): Switch to next tab for this window

app.tab\_open(): Open a new tab

app.tab\_previous(): Switch to previous tab for this window

app.tab\_reopen(): Re-open the last-closed tab

app.window\_close(): Close the current window

app.window\_hide(): Hide the current window

app.window\_hide\_others(): Hide all other windows

app.window\_next(): Switch to next window for this app

app.window\_open(): Open a new window

app.window\_previous(): Switch to previous window for this app

browser.address() -> str: Get page URL

browser.bookmark(): Bookmark the current page

browser.bookmark\_tabs(): Bookmark all open tabs

browser.bookmarks(): Open the Bookmarks editor

browser.bookmarks\_bar(): Toggle the bookmarks bar

browser.focus\_address(): Focus address bar

browser.focus\_page(): Focus the page body

browser.focus\_search(): Focus the search box

browser.go(url: str): Go to a new URL

browser.go\_back(): Go back in the history

browser.go\_blank(): Go to a blank page

browser.go\_forward(): Go forward in the history

browser.go\_home(): Go to home page

browser.open\_private\_window(): Open a private browsing window

browser.reload(): Reload current page

browser.reload\_hard(): Reload current page (harder)

browser.reload\_hardest(): Reload current page (hardest)

browser.show\_clear\_cache(): Show 'Clear Cache' dialog

browser.show\_downloads(): Show download list

browser.show\_extensions(): Show installed extensions

browser.show\_history(): Show recently visited pages

browser.submit\_form(): Submit the current form

browser.title() -> str: Get page title

browser.toggle\_dev\_tools(): Open or close the developer tools

clip.capture\_text(key: str): Send key sequence and return resulting clipboard text

clip.clear() -> None: Clear clipboard contents

clip.image() -> Optional[talon.skia.image.Image]: Get clipboard image

clip.set\_image(image: talon.skia.image.Image): Set clipboard image

clip.set\_text(text: str): Set clipboard text

clip.text() -> str: Get clipboard text

code.complete(): Trigger code autocomplete

code.extend\_scope\_end(): Extend selection to end of current scope

code.extend\_scope\_in(): Extend selection to start of first inner scope

code.extend\_scope\_next(): Extend selection to start of next sibling scope

code.extend\_scope\_out(): Extend selection to start of outer scope

code.extend\_scope\_previous(): Extend selection to start of previous sibling scope

code.extend\_scope\_start(): Extend selection to start of current scope

code.language() -> str: Return the active programming language

code.rename(name: str): Rename selection to <name>

code.scope\_end(): Move cursor to end of current scope

code.scope\_in(): Move cursor to start of first inner scope

code.scope\_next(): Move cursor to start of next sibling scope

code.scope\_out(): Move cursor to start of outer scope

code.scope\_previous(): Move cursor to start of previous sibling scope

code.scope\_start(): Move cursor to start of current scope

code.select\_scope(): Select scope under cursor

code.toggle\_comment(): Toggle comments on the current line(s)

core.cancel\_phrase\_\_unstable(): Cancel the currently running phrase

core.current\_command\_\_unstable() -> tuple[talon.scripting.types.CommandImpl, talon.grammar.vm.Capture]: Return the currently executing command

core.last\_command() -> tuple[talon.scripting.types.CommandImpl, talon.grammar.vm.Capture]: Return the last executed command

core.last\_phrase() -> talon.grammar.vm.Capture: Return the last-spoken phrase

core.recent\_commands() -> Sequence[Sequence[tuple[talon.scripting.types.CommandImpl, talon.grammar.vm.Capture]]]: Return recently executed commands (grouped by phrase)

core.recent\_phrases() -> Sequence[talon.grammar.vm.Capture]: Return recently-spoken phrases

core.repeat\_command(times: int = 1): Repeat the last command N times

core.repeat\_partial\_phrase(times: int = 1): Repeat the previous phrase or current partial phrase N times

core.repeat\_phrase(times: int = 1): Repeat the last phrase N times

core.replace\_command(commands: Sequence[tuple[talon.scripting.types.CommandImpl, talon.grammar.vm.Capture]]): Replace the current command in history with one or more commands

core.run\_command(cmd: talon.scripting.types.CommandImpl, m: talon.grammar.vm.Capture): Run a single command for a recognized phrase

core.run\_hotkey(hotkey: talon.scripting.types.ScriptImpl): Run all commands for a hotkey

core.run\_phrase(phrase: talon.grammar.vm.Capture): Run all commands for a recognized phrase

core.run\_talon\_script(ctx: talon.scripting.rctx.ResourceContext, script: talon.scripting.talon\_script.TalonScript, m: talon.grammar.vm.Capture): Run a single TalonScript for a recognized phrase

dictate.join\_words(words: Sequence[str], separator: str = ' ') -> str: Join a list of words into a single string for insertion

dictate.lower(p: talon.grammar.vm.Phrase): Insert lowercase text with auto\_insert()

dictate.natural(p: talon.grammar.vm.Phrase): Insert naturally-capitalized text with auto\_insert()

dictate.parse\_words(p: talon.grammar.vm.Phrase) -> Sequence[str]: Extract words from a spoken Capture

dictate.replace\_words(words: Sequence[str]) -> Sequence[str]: Replace words according to the dictate.word\_map dictionary setting

edit.copy(): Copy selection to clipboard

edit.cut(): Cut selection to clipboard

edit.delete(): Delete selection

edit.delete\_line(): Delete line under cursor

edit.delete\_paragraph(): Delete paragraph under cursor

edit.delete\_sentence(): Delete sentence under cursor

edit.delete\_word(): Delete word under cursor

edit.down(): Move cursor down one row

edit.extend\_again(): Extend selection again in the same way

edit.extend\_column(n: int): Extend selection to column <n>

edit.extend\_down(): Extend selection down one row

edit.extend\_file\_end(): Extend selection to end of file

edit.extend\_file\_start(): Extend selection to start of file

edit.extend\_left(): Extend selection left one column

edit.extend\_line(n: int): Extend selection to include line <n>

edit.extend\_line\_down(): Extend selection down one full line

edit.extend\_line\_end(): Extend selection to end of line

edit.extend\_line\_start(): Extend selection to start of line

edit.extend\_line\_up(): Extend selection up one full line

edit.extend\_page\_down(): Extend selection down one page

edit.extend\_page\_up(): Extend selection up one page

edit.extend\_paragraph\_end(): Extend selection to the end of the current paragraph

edit.extend\_paragraph\_next(): Extend selection to the start of the next paragraph

edit.extend\_paragraph\_previous(): Extend selection to the start of the previous paragraph

edit.extend\_paragraph\_start(): Extend selection to the start of the current paragraph

edit.extend\_right(): Extend selection right one column

edit.extend\_sentence\_end(): Extend selection to the end of the current sentence

edit.extend\_sentence\_next(): Extend selection to the start of the next sentence

edit.extend\_sentence\_previous(): Extend selection to the start of the previous sentence

edit.extend\_sentence\_start(): Extend selection to the start of the current sentence

edit.extend\_up(): Extend selection up one row

edit.extend\_word\_left(): Extend selection left one word

edit.extend\_word\_right(): Extend selection right one word

edit.file\_end(): Move cursor to end of file (start of line)

edit.file\_start(): Move cursor to start of file

edit.find(text: str = None): Open Find dialog, optionally searching for text

edit.find\_next(): Select next Find result

edit.find\_previous(): Select previous Find result

edit.indent\_less(): Remove a tab stop of indentation

edit.indent\_more(): Add a tab stop of indentation

edit.jump\_column(n: int): Move cursor to column <n>

edit.jump\_line(n: int): Move cursor to line <n>

edit.left(): Move cursor left one column

edit.line\_clone(): Create a new line identical to the current line

edit.line\_down(): Move cursor to start of line below

edit.line\_end(): Move cursor to end of line

edit.line\_insert\_down(): Insert line below cursor

edit.line\_insert\_up(): Insert line above cursor

edit.line\_start(): Move cursor to start of line

edit.line\_swap\_down(): Swap the current line with the line below

edit.line\_swap\_up(): Swap the current line with the line above

edit.line\_up(): Move cursor to start of line above

edit.move\_again(): Move cursor again in the same way

edit.page\_down(): Move cursor down one page

edit.page\_up(): Move cursor up one page

edit.paragraph\_end(): Move cursor to the end of the current paragraph

edit.paragraph\_next(): Move cursor to the start of the next paragraph

edit.paragraph\_previous(): Move cursor to the start of the previous paragraph

edit.paragraph\_start(): Move cursor to the start of the current paragraph

edit.paste(): Paste clipboard at cursor

edit.paste\_match\_style(): Paste clipboard without style information

edit.print(): Open print dialog

edit.redo(): Redo

edit.right(): Move cursor right one column

edit.save(): Save current document

edit.save\_all(): Save all open documents

edit.select\_all(): Select all text in the current document

edit.select\_line(n: int = None): Select entire line <n>, or current line

edit.select\_lines(a: int, b: int): Select entire lines from <a> to <b>

edit.select\_none(): Clear current selection

edit.select\_paragraph(): Select the entire nearest paragraph

edit.select\_sentence(): Select the entire nearest sentence

edit.select\_word(): Select word under cursor

edit.selected\_text() -> str: Get currently selected text

edit.selection\_clone(): Insert a copy of the current selection

edit.sentence\_end(): Move cursor to the end of the current sentence

edit.sentence\_next(): Move cursor to the start of the next sentence

edit.sentence\_previous(): Move cursor to the start of the previous sentence

edit.sentence\_start(): Move cursor to the start of the current sentence

edit.undo(): Undo

edit.up(): Move cursor up one row

edit.word\_left(): Move cursor left one word

edit.word\_right(): Move cursor right one word

edit.zoom\_in(): Zoom in

edit.zoom\_out(): Zoom out

edit.zoom\_reset(): Zoom to original size

auto\_format(text: str) -> str: Apply text formatting, such as auto spacing, for the native language

auto\_insert(text: str): Insert text at the current cursor position, automatically formatting it using the actions.auto\_format(text)

insert(text: str): Insert text at the current cursor position

key(key: str): Press one or more keys by name, space-separated

mimic(text: str): Simulate speaking {text}

mouse\_click(button: int = 0): Press and release a mouse button

mouse\_drag(button: int = 0): Hold down a mouse button

mouse\_move(x: float, y: float): Move mouse to (x, y) coordinate

mouse\_release(button: int = 0): Release a mouse button

mouse\_scroll(y: float = 0, x: float = 0, by\_lines: bool = False): Scroll the mouse wheel

mouse\_x() -> float: Mouse X position

mouse\_y() -> float: Mouse Y position

print(obj: Any): Display an object in the log

skip(): Do nothing

sleep(duration: Union[float, str]): Pause for some duration.: If you use a number, it is seconds, e.g 1.5 seconds or 0.001 seconds.: If you use a string, it is a timespec, such as "50ms" or "10s": For performance reasons, sleep() cannot be reimplemented by a Context.

migrate.backup\_user(): Backup the .talon/user/ directory to a zip file in .talon/backups/

migrate.v02\_all(prefix: str = '', verbose: bool = False): Perform migrations for Talon v0.2 on all files in user/

migrate.v02\_one(path: str, verbose: bool = False): Migrate action() definitions from a .talon file to a new Python file.

mode.disable(mode: str): Disable a mode

mode.enable(mode: str): Enable a mode

mode.restore(): Restore saved modes

mode.save(): Save all active modes

mode.toggle(mode: str): Toggle a mode

path.talon\_app() -> str: Path to Talon application

path.talon\_home() -> str: Path to home/.talon

path.talon\_user() -> str: Path to Talon user

path.user\_home() -> str: Path to user home

speech.disable(): Disable speech recognition

speech.enable(): Enable speech recognition

speech.enabled() -> bool: Test if speech recognition is enabled

speech.record\_flac(): Record the phrase audio to a flac file

speech.record\_wav(): Record the phrase audio to a wave file

speech.replay(path: str): Replay a .flac or .wav file into the speech engine

speech.set\_microphone(name: str): Set the currently active microphone - DEPRECATED: use sound.set\_microphone()

speech.toggle(value: bool = None): Toggle speech recognition

sound.active\_microphone() -> str: Return active microphone name

sound.microphones() -> Sequence[str]: Return a list of available microphone names

sound.set\_microphone(name: str): Set the currently active microphone

win.file\_ext() -> str: Return the open file's extension

win.filename() -> str: Return the open filename

win.title() -> str: Get window title

tracking.calibrate() -> None: Calibrate Eye Tracking

tracking.control1\_enabled() -> bool: Is Control Mouse (Legacy) Enabled?

tracking.control1\_toggle(state: bool = None) -> None: Toggle Control Mouse (Legacy)

tracking.control\_debug\_toggle(state: bool = None) -> None: Toggle Control Mouse 2 (Debug View)

tracking.control\_enabled() -> bool: Is Control Mouse Enabled?

tracking.control\_gaze\_focus\_toggle(state: bool = None) -> None: Toggle Control Mouse 2 (Gaze Focus)

tracking.control\_gaze\_scroll\_toggle(state: bool = None) -> None: Toggle Control Mouse 2 (Gaze Scroll)

tracking.control\_gaze\_toggle(state: bool = None) -> None: Toggle Control Mouse 2 (Gaze Control)

tracking.control\_head\_toggle(state: bool = None) -> None: Toggle Control Mouse 2 (Head Control)

tracking.control\_mouse\_jump\_toggle(state: bool = None) -> None: Toggle Control Mouse 2 (Mouse Jump)

tracking.control\_toggle(state: bool = None) -> None: Toggle Control Mouse

tracking.control\_zoom\_enabled() -> bool: Is Control Mouse (Zoom) Enabled?

tracking.control\_zoom\_toggle(state: bool = None) -> None: Toggle Control Mouse (Zoom)

tracking.zoom() -> None: Trigger Eye Zoom / Click

tracking.zoom\_cancel() -> None: Cancel Eye Zoom

menu.check\_for\_updates(): Check for updates

menu.open\_debug\_window(): Open Debug window

menu.open\_log(): Open Talon log

menu.open\_repl(): Open Talon REPL

menu.open\_talon\_home(): Open Talon config folder

user.cheatsheet(): Print out a sheet of talon commands

user.password\_delete(): Delete password

user.password\_duplicate(): Duplicate password

user.password\_edit(): Edit password

user.password\_fill(): fill the password

user.password\_new(): New password

user.password\_show(): show the password

user.chrome\_mod(key: str): Press the specified key with the correct modifier key for the OS

user.discord\_answer\_call(): Answer incoming call

user.discord\_deafen(): Toggle deafen

user.discord\_decline\_call(): Decline incoming call

user.discord\_emoji\_picker(): Toggle emoji picker

user.discord\_gif\_picker(): Toggle gif picker

user.discord\_go\_current\_call(): Go to current call

user.discord\_mark\_inbox\_read(): Mark top inbox channel read

user.discord\_mentions\_last(): Go up to channel with unread mentions

user.discord\_mentions\_next(): Go down to channel with unread mentions

user.discord\_mute(): Toggle mute

user.discord\_oldest\_unread(): Go to oldest unread message

user.discord\_quick\_switcher(dest\_type: str, dest\_search: str): Open up the quick switcher, optionally specifying a type of destination

user.discord\_toggle\_dms(): Toggle between dms and your most recent server

user.discord\_toggle\_inbox(): Toggle inbox popout

user.discord\_toggle\_members(): Toggle channel member list

user.discord\_toggle\_pins(): Toggle pins popout

user.i3wm\_launch(): Trigger the i3 launcher: ex rofi

user.i3wm\_lock(): Trigger the lock screen

user.i3wm\_shell(): Launch a shell

user.idea(commands: str): Send a command to Jetbrains product

user.idea\_grab(times: int): Copies specified number of words to the left

user.thunderbird\_calendar\_view(number: int): Select between calendar view tabs

user.thunderbird\_mod(keys: str): Press keys with modifier ctrl or cmd

user.tmux\_enter\_command(command: str = ''): Enter tmux command mode and optionally insert a command without executing it.

user.tmux\_execute\_command(command: str): execute tmux command

user.tmux\_execute\_command\_with\_confirmation(command: str, confirmation\_prompt: str): execute tmux command with confirm-before

user.tmux\_keybind(key: str): press tmux prefix followed by a key bind

user.tmux\_prefix(): press control and the configured tmux prefix key

user.command\_server\_directory() -> str: The dirctory which contains the files required for communication between: the application and Talon. This is the only function which absolutely: must be implemented for any application using the command-client. Each: application that supports file-based RPC should use its own directory: name. Note that this action should only return a name; the parent: directory is determined by the core command client code.

user.did\_emit\_pre\_phrase\_signal() -> bool: Indicates whether the pre-phrase signal was emitted at the start of this phrase

user.emit\_pre\_phrase\_signal() -> bool: If in an application supporting the command client, returns True: and touches a file to indicate that a phrase is beginning execution.: Otherwise does nothing and returns False.

user.run\_rpc\_command(command\_id: str, arg1: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg2: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg3: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg4: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg5: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>): Execute command via RPC.

user.run\_rpc\_command\_and\_wait(command\_id: str, arg1: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg2: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg3: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg4: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg5: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>): Execute command via application command server and wait for command to finish.

user.run\_rpc\_command\_get(command\_id: str, arg1: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg2: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg3: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg4: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg5: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>) -> Any: Execute command via application command server and return command output.

user.trigger\_command\_server\_command\_execution(): Issue keystroke to trigger command server to execute command that: was written to the file. For internal use only

user.vscode(command\_id: str): Execute command via vscode command server, if available, or fallback: to command palette.

user.vscode\_and\_wait(command\_id: str): Execute command via vscode command server, if available, and wait: for command to finish. If command server not available, uses command: palette and doesn't guarantee that it will wait for command to: finish.

user.vscode\_get(command\_id: str, arg1: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg2: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg3: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg4: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg5: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>) -> Any: Execute command via vscode command server and return command output.

user.vscode\_with\_plugin(command\_id: str, arg1: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg2: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg3: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg4: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg5: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>): Execute command via vscode command server.

user.vscode\_with\_plugin\_and\_wait(command\_id: str, arg1: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg2: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg3: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg4: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>, arg5: Any = <class 'user.knausj\_talon.apps.vscode.command\_client.command\_client.NotSet'>): Execute command via vscode command server and wait for command to finish.

user.command\_palette(): Show command palette

user.vscode\_terminal(number: int): Activate a terminal by number

user.wsl\_reset\_path\_detection(): reset wsl path detection

user.wsl\_speak(): ask each distro to say hello (in the log)

user.get\_running\_app(name: str) -> talon.windows.ui.App: Get the first available running app with `name`.

user.switcher\_focus(name: str): Focus a new application by name

user.switcher\_focus\_app(app: talon.windows.ui.App): Focus application and wait until switch is made

user.switcher\_focus\_window(window: talon.windows.ui.Window): Focus window and wait until switch is made

user.switcher\_hide\_running(): Hides list of running applications

user.switcher\_launch(path: str): Launch a new application by path (all OSes), or AppUserModel\_ID path on Windows

user.switcher\_menu(): Open a menu of running apps to switch to

user.switcher\_toggle\_running(): Shows/hides all running applications

user.delayed\_speech\_off(): Disables "temporary speech" mode lazily, meaning that the next: phrase that finishes will turn speech off.

user.delayed\_speech\_on(): Activates a "temporary speech" mode that can be disabled lazily,: so that the actual disable command happens after whatever phrase: finishes next.

user.deprecate\_action(time\_deprecated: str, name: str): Notify the user that the given action is deprecated and should: not be used into the future.

user.deprecate\_capture(time\_deprecated: str, name: str): Notify the user that the given capture is deprecated and should: not be used into the future.

user.deprecate\_command(time\_deprecated: str, name: str, replacement: str): Notify the user that the given voice command is deprecated and should: not be used into the future; the command `replacement` should be used: instead.

user.copy\_word\_left(): Copies the word to the left.

user.copy\_word\_right(): Copies the word to the right.

user.cut\_line(): Cuts the current line.

user.cut\_word\_left(): Cuts the word to the left.

user.cut\_word\_right(): Cuts the word to the right.

user.paste(text: str): Pastes text and preserves clipboard

user.words\_left(n: int): Moves left by n words.

user.words\_right(n: int): Moves right by n words.

user.insert\_between(before: str, after: str): Insert `before + after`, leaving cursor between `before` and `after`. Not entirely reliable if `after` contains newlines.

user.insert\_cursor(text: str): Insert a string. Leave the cursor wherever [|] is in the text

user.edit\_text\_file(path: str): Tries to open a file in the user's preferred text editor.

user.engine\_mimic(cmd: str): Sends phrase to engine

user.engine\_sleep(): Sleep the engine

user.engine\_wake(): Wake the engine

user.help\_context(): Display contextual command info

user.help\_context\_enabled(): Display contextual command info

user.help\_formatters(ab: dict): Provides the list of formatter keywords

user.help\_hide(): Hides the help

user.help\_list(ab: str): Provides the symbol dictionary

user.help\_next(): Navigates to next page

user.help\_previous(): Navigates to previous page

user.help\_refresh(): Refreshes the help

user.help\_return(): Returns to the main help window

user.help\_search(phrase: str): Display command info for search phrase

user.help\_select\_index(index: int): Select the context by a number

user.help\_selected\_context(m: str): Display command info for selected context

user.help\_scope\_toggle(): Toggle help scope gui

user.homophones\_force\_show(m: str): Show the homophones display forcibly

user.homophones\_force\_show\_selection(): Show the homophones display for the selected text forcibly

user.homophones\_get(word: str) -> [<class 'str'>]: Get homophones for the given word

user.homophones\_hide(): Hides the homophones display

user.homophones\_select(number: int) -> str: selects the homophone by number

user.homophones\_show(m: str): Show the homophones display

user.homophones\_show\_auto(): Show homophones for selection, or current word if selection is empty.

user.homophones\_show\_selection(): Show the homophones display for the selected text

user.code\_clear\_language\_mode(): Clears the active language mode, and re-enables code.language: extension matching

user.code\_set\_language\_mode(language: str): Sets the active language mode, and disables extension matching

user.dragon\_mode(): For windows and Mac with Dragon, disables Talon commands and exits Dragon's command mode

user.talon\_mode(): For windows and Mac with Dragon, enables Talon commands and Dragon's command mode.

user.grid\_activate(): Show mouse grid

user.grid\_close(): Close the active grid

user.grid\_go\_back(): Sets the grid state back to what it was before the last command

user.grid\_narrow(digit: Union[int, str]): Choose a field of the grid and narrow the selection down

user.grid\_narrow\_list(digit\_list: list[str]): Choose fields multiple times in a row

user.grid\_place\_window(): Places the grid on the currently active window

user.grid\_reset(): Resets the grid to fill the whole screen again

user.grid\_select\_screen(screen: int): Brings up mouse grid

user.screens\_get\_by\_number(screen\_number: int) -> talon.screen.Screen: Get screen by number

user.screens\_get\_next(screen: talon.screen.Screen) -> talon.screen.Screen: Get the screen after this one

user.screens\_get\_previous(screen: talon.screen.Screen) -> talon.screen.Screen: Get the screen before this one

user.screens\_show\_numbering(): Show screen number on each screen

user.system\_command(cmd: str): execute a command on the system

user.system\_command\_nb(cmd: str): execute a command on the system without blocking

user.add\_phrase\_to\_history(text: str): Adds a phrase to the phrase history

user.before\_last\_phrase(): Moves left before the last phrase

user.clear\_last\_phrase(): Clears the last phrase

user.get\_last\_phrase() -> str: Gets the last phrase

user.get\_recent\_phrase(number: int) -> str: Gets the nth most recent phrase

user.phrase\_history\_hide(): Hides the recent phrases window

user.select\_last\_phrase(): Selects the last phrase

user.toggle\_phrase\_history(): Toggles list of recent phrases

user.dictation\_format\_cap(): Sets the dictation formatter to capitalize

user.dictation\_format\_no\_cap(): Sets the dictation formatter to not capitalize

user.dictation\_format\_no\_space(): Sets the dictation formatter to not prepend a space

user.dictation\_format\_reset(): Resets the dictation formatter

user.dictation\_insert(text: str, auto\_cap: bool = True) -> str: Inserts dictated text, formatted appropriately.

user.dictation\_insert\_raw(text: str): Inserts text as-is, without invoking the dictation formatter.

user.dictation\_peek(left: bool, right: bool) -> tuple[typing.Optional[str], typing.Optional[str]]: Gets text around the cursor to inform auto-spacing and -capitalization.: Returns (before, after), where `before` is some text before the cursor,: and `after` some text after it. Results are not guaranteed; `before`: and/or `after` may be None, indicating no information. If `before` is: the empty string, this means there is nothing before the cursor (we are: at the beginning of the document); likewise for `after`.

: To optimize performance, pass `left = False` if you won't need: `before`, and `right = False` if you won't need `after`.

: dictation\_peek() is intended for use before inserting text, so it may: delete any currently selected text.

user.dictation\_reformat\_cap(): Capitalizes the last utterance

user.dictation\_reformat\_no\_cap(): Lowercases the last utterance

user.dictation\_reformat\_no\_space(): Removes space before the last utterance

user.open\_url(url: str): Visit the given URL.

user.search\_with\_search\_engine(search\_template: str, search\_text: str): Search a search engine for given text

user.tab\_close\_wrapper(): Closes the current tab.: Exists so that apps can implement their own delay before running tab\_close() to handle repetitions better.

user.tab\_duplicate(): Duplicates the current tab.

user.tab\_final(): Jumps to the final tab

user.tab\_jump(number: int): Jumps to the specified tab

user.move\_app\_to\_screen(app\_name: str, screen\_number: int): Move a specific application to another screen.

user.move\_window\_next\_screen() -> None: Move the active window to a specific screen.

user.move\_window\_previous\_screen() -> None: Move the active window to the previous screen.

user.move\_window\_to\_screen(screen\_number: int) -> None: Move the active window leftward by one.

user.snap\_app(app\_name: str, pos: user.knausj\_talon.core.windows\_and\_tabs.window\_snap.RelativeScreenPos): Snap a specific application to another screen.

user.snap\_window(pos: user.knausj\_talon.core.windows\_and\_tabs.window\_snap.RelativeScreenPos) -> None: Move the active window to a specific position on-screen.

: See `RelativeScreenPos` for the structure of this position.

user.code\_comment\_documentation\_block(): Inserts a block document comment and positions the cursor appropriately

user.code\_comment\_documentation\_block\_inner(): Inserts an inner block document comment and positions the cursor appropriately

user.code\_comment\_documentation\_inner(): Inserts an inner document comment and positions the cursor appropriately

user.code\_insert\_if\_let\_error(): Inserts if let error block, positioning the cursor appropriately

user.code\_insert\_if\_let\_some(): Inserts if let some block, positioning the cursor appropriately

user.code\_insert\_macro(text: str, selection: str): Inserts a macro and positions the cursor appropriately

user.code\_insert\_macro\_array(text: str, selection: str): Inserts a macro array and positions the cursor appropriately

user.code\_insert\_macro\_block(text: str, selection: str): Inserts a macro block and positions the cursor appropriately

user.code\_insert\_return\_trait(type: str): Inserts a return type for implementor of trait

user.code\_insert\_trait\_annotation(type: str): Inserts type annotation for implementor of trait

user.code\_state\_implements(): Inserts implements block, positioning the cursor appropriately

user.code\_state\_unsafe(): Inserts an unsafe block and positions the cursor appropriately

user.code\_comment\_block(): Block comment

user.code\_comment\_block\_prefix(): Block comment start syntax

user.code\_comment\_block\_suffix(): Block comment end syntax

user.code\_comment\_documentation(): Inserts a document comment and positions the cursor appropriately

user.code\_comment\_line\_prefix(): Inserts line comment prefix at current cursor location

user.code\_insert\_false(): Insert False value

user.code\_insert\_true(): Insert True value

user.code\_insert\_is\_not\_null(): Inserts check for non-null

user.code\_insert\_is\_null(): Inserts check for null

user.code\_insert\_null(): Inserts null

user.code\_default\_function(text: str): Inserts function declaration

user.code\_insert\_named\_argument(parameter\_name: str): Inserts a named argument

user.code\_insert\_return\_type(type: str): Inserts a return type

user.code\_insert\_type\_annotation(type: str): Inserts a type annotation

user.code\_modified\_function(modifiers: Union[list[str], int], text: str): Inserts function declaration with the given modifiers. modifiers == 0: implies no modifiers (.talon files don't have empty list literal: syntax)

user.code\_private\_function(text: str): Inserts private function declaration

user.code\_private\_function\_formatter(name: str): Inserts private function name with formatter

user.code\_private\_static\_function(text: str): Inserts private static function

user.code\_private\_variable\_formatter(name: str): inserts properly formatted private function name

user.code\_protected\_function(text: str): Inserts protected function declaration

user.code\_protected\_function\_formatter(name: str): inserts properly formatted private function name

user.code\_protected\_static\_function(text: str): Inserts public function

user.code\_protected\_variable\_formatter(name: str): inserts properly formatted private function name

user.code\_public\_function(text: str): Inserts public function

user.code\_public\_function\_formatter(name: str): inserts properly formatted private function name

user.code\_public\_static\_function(text: str): Inserts public function

user.code\_public\_variable\_formatter(name: str): inserts properly formatted private function name

user.code\_insert\_function(text: str, selection: str): Inserts a function and positions the cursor appropriately

user.code\_select\_function(number: int, selection: str): Inserts the selected function when the imgui is open

user.code\_toggle\_functions(): GUI: List functions for active language

user.code\_block(): Inserts equivalent of {

} for the active language, and places the cursor appropriately

user.code\_break(): Inserts break statement

user.code\_next(): Inserts next statement

user.code\_state\_case(): Inserts case statement

user.code\_state\_do(): Inserts do statement

user.code\_state\_else(): Inserts else statement

user.code\_state\_else\_if(): Inserts else if statement

user.code\_state\_for(): Inserts for statement

user.code\_state\_for\_each(): Inserts for each equivalent statement

user.code\_state\_go\_to(): inserts go-to statement

user.code\_state\_if(): Inserts if statement

user.code\_state\_infinite\_loop(): Inserts infinite loop statement

user.code\_state\_return(): Inserts return statement

user.code\_state\_switch(): Inserts switch statement

user.code\_state\_while(): Inserts while statement

user.code\_try\_catch(): Inserts try/catch. If selection is true, does so around the selection

user.code\_keyword(keywords: list[str]): Adds keywords

user.code\_import(): import/using equivalent

user.code\_insert\_library(text: str, selection: str): Inserts a library and positions the cursor appropriately

user.code\_select\_library(number: int, selection: str): Inserts the selected library when the imgui is open

user.code\_toggle\_libraries(): GUI: List libraries for active language

user.code\_define\_class(): Starts a class definition (e.g., Java's "class" keyword)

user.code\_operator\_object\_accessor(): Inserts the object accessor operator (e.g., Java's "." or PHP's "->)

user.code\_self(): Inserts a reference to the current object (e.g., C++ "this" or Python's "self")

user.code\_operator\_subscript(): code\_operator\_subscript (e.g., C++ [])

user.code\_operator\_addition\_assignment(): code\_operator\_addition\_assignment

user.code\_operator\_assignment(): code\_operator\_assignment

user.code\_operator\_bitwise\_and\_assignment(): code\_operator\_and\_assignment

user.code\_operator\_bitwise\_exclusive\_or\_assignment(): code\_operator\_bitwise\_exclusive\_or\_assignment

user.code\_operator\_bitwise\_left\_shift\_assignment(): code\_operator\_bitwise\_left\_shift\_assigment

user.code\_operator\_bitwise\_or\_assignment(): code\_operator\_or\_assignment

user.code\_operator\_bitwise\_right\_shift\_assignment(): code\_operator\_bitwise\_right\_shift\_assignment

user.code\_operator\_division\_assignment(): code\_operator\_division\_assignment

user.code\_operator\_increment(): code\_operator\_increment

user.code\_operator\_modulo\_assignment(): code\_operator\_modulo\_assignment

user.code\_operator\_multiplication\_assignment(): code\_operator\_multiplication\_assignment

user.code\_operator\_subtraction\_assignment(): code\_operator\_subtraction\_assignment

user.code\_operator\_bitwise\_and(): code\_operator\_bitwise\_and

user.code\_operator\_bitwise\_exclusive\_or(): code\_operator\_bitwise\_exclusive\_or

user.code\_operator\_bitwise\_left\_shift(): code\_operator\_bitwise\_left\_shift

user.code\_operator\_bitwise\_or(): code\_operator\_bitwise\_or

user.code\_operator\_bitwise\_right\_shift(): code\_operator\_bitwise\_right\_shift

user.code\_operator\_lambda(): code\_operator\_lambda

user.code\_operator\_addition(): code\_operator\_addition

user.code\_operator\_and(): code\_operator\_and

user.code\_operator\_division(): code\_operator\_division

user.code\_operator\_equal(): code\_operator\_equal

user.code\_operator\_exponent(): code\_operator\_exponent

user.code\_operator\_greater\_than(): code\_operator\_greater\_than

user.code\_operator\_greater\_than\_or\_equal\_to(): code\_operator\_greater\_than\_or\_equal\_to

user.code\_operator\_in(): code\_operator\_in

user.code\_operator\_less\_than(): code\_operator\_less\_than

user.code\_operator\_less\_than\_or\_equal\_to(): code\_operator\_less\_than\_or\_equal\_to

user.code\_operator\_modulo(): code\_operator\_modulo

user.code\_operator\_multiplication(): code\_operator\_multiplication

user.code\_operator\_not\_equal(): code\_operator\_not\_equal

user.code\_operator\_not\_in(): code\_operator\_not\_in

user.code\_operator\_or(): code\_operator\_or

user.code\_operator\_subtraction(): code\_operator\_subtraction

user.code\_operator\_address\_of(): code\_operator\_address\_of (e.g., C++ & op)

user.code\_operator\_indirection(): code\_operator\_indirection

user.code\_operator\_structure\_dereference(): code\_operator\_structure\_dereference (e.g., C++ -> op)

user.code\_terraform\_data\_source(text: str): Inserts a new data block with given name

user.code\_terraform\_module\_block(text: str): Inserts a new module-related block of a given type (e.g. variable, output, provider...)

user.code\_terraform\_resource(text: str): Inserts a new resource block with given name

user.history\_clear(): Clear the history

user.history\_disable(): Disables the history

user.history\_enable(): Enables the history

user.history\_get(number: int) -> str: returns the history entry at the specified index

user.history\_less(): Show less history

user.history\_more(): Show more history

user.history\_toggle(): Toggles viewing the history

user.history\_transform\_phrase\_text(words: list[str]) -> Optional[str]: Transforms phrase text for presentation in history. Return `None` to omit from history

user.time\_format(fmt: str = None) -> str: Return the current time, formatted.: fmt: strftime()-style format string, defaults to ISO format.

user.time\_format\_utc(fmt: str = None) -> str: Return the current UTC time, formatted.: fmt: strftime()-style format string, defaults to ISO format.

user.desktop(number: int): change the current desktop

user.desktop\_last(): move to previous desktop

user.desktop\_next(): move to next desktop

user.desktop\_show(): shows the current desktops

user.window\_move\_desktop(desktop\_number: int): move the current window to a different desktop

user.window\_move\_desktop\_left(): move the current window to the desktop to the left

user.window\_move\_desktop\_right(): move the current window to the desktop to the right

user.draft\_editor\_discard(): Discard draft editor

user.draft\_editor\_open(): Open draft editor

user.draft\_editor\_paste\_last(): Paste last submitted draft

user.draft\_editor\_submit(): Submit/save draft editor

user.macro\_append\_command(words: list[str]): Appends a command to the current macro; called when a voice command is uttered while recording a macro.

user.macro\_copy(name: str): Copied the specified macro to the clipboard as a Talon command.

user.macro\_list(): List all saved macros.

user.macro\_list\_close(): Closed the saved macros list.

user.macro\_play(name: str): Execute the commands in the last recorded macro.

user.macro\_record(): Begin recording a new voice command macro.

user.macro\_save(name: str): Save the macro.

user.macro\_stop(): Stop recording the macro.

user.play\_pause(): Plays or pauses media

user.microphone\_select(index: int): Selects a micropohone

user.microphone\_selection\_hide(): Hide the microphone selection GUI

user.microphone\_selection\_toggle(): Show GUI for choosing the Talon microphone

user.copy\_mouse\_position(): Copy the current mouse position coordinates

user.mouse\_drag(button: int): Press and hold/release a specific mouse button for dragging

user.mouse\_drag\_end(): Releases any held mouse buttons

user.mouse\_gaze\_scroll(): Starts gaze scroll

user.mouse\_hide\_cursor(): Hides the cursor

user.mouse\_move\_center\_active\_window(): move the mouse cursor to the center of the currently active window

user.mouse\_scroll\_down(amount: float = 1): Scrolls down

user.mouse\_scroll\_down\_continuous(): Scrolls down continuously

user.mouse\_scroll\_left(amount: float = 1): Scrolls left

user.mouse\_scroll\_right(amount: float = 1): Scrolls right

user.mouse\_scroll\_stop(): Stops scrolling

user.mouse\_scroll\_up(amount: float = 1): Scrolls up

user.mouse\_scroll\_up\_continuous(): Scrolls up continuously

user.mouse\_show\_cursor(): Shows the cursor

user.mouse\_sleep(): Disables control mouse, zoom mouse, and re-enables cursor

user.mouse\_wake(): Enable control mouse, zoom mouse, and disables cursor

user.screenshot(screen\_number: Optional[int] = None): Takes a screenshot of the entire screen and saves it to the pictures folder.: Optional screen number can be given to use screen other than main.

user.screenshot\_clipboard(screen\_number: Optional[int] = None): Takes a screenshot of the entire screen and saves it to the clipboard.: Optional screen number can be given to use screen other than main.

user.screenshot\_selection(): Triggers an application that is capable of taking a screenshot of a portion of the screen

user.screenshot\_settings(): Opens the settings UI for screenshots.: Only applies to Mac for now

user.screenshot\_window(): Takes a screenshot of the active window and saves it to the pictures folder

user.screenshot\_window\_clipboard(): Takes a screenshot of the active window and saves it to the clipboard

user.draft\_get\_text() -> str: Returns the text in the draft window

user.draft\_hide(): Hides draft window

user.draft\_named\_move(name: str, screen\_number: Optional[int] = None): Lets you move the window to the top, bottom, left, right, or middle: of the screen.

user.draft\_position\_caret(anchor: str, after: int = 0): Positions the caret in the draft window

user.draft\_resize(width: int, height: int): Resize the draft window.

user.draft\_select(start\_anchor: str, end\_anchor: str = '', include\_trailing\_whitespace: int = 0): Selects text in the draft window

user.draft\_show(text: Optional[str] = None): Shows draft window

user.talon\_action\_find(action: str): Runs action.find for the provided action and dumps to the log

user.talon\_add\_context\_clipboard(): Adds os-specific context info to the clipboard for the focused app for .talon files

user.talon\_add\_context\_clipboard\_python(): Adds os-specific context info to the clipboard for the focused app for .py files. Assumes you've a Module named mod declared.

user.talon\_copy\_list(name: str): Dumps the contents of list to the console

user.talon\_debug\_all\_settings(): Dumps all settings to the console

user.talon\_debug\_app\_windows(app: str): Pretty prints the application windows

user.talon\_debug\_list(name: str): Dumps the contents of list to the console

user.talon\_debug\_modes(): Dumps active modes to the console

user.talon\_debug\_scope(name: str): Dumps the active scope information to the console

user.talon\_debug\_setting(name: str): Dumps the current setting to the console

user.talon\_debug\_tags(): Dumps the active tags to the console

user.talon\_get\_active\_application\_info() -> str: Returns all active app info to the cliboard

user.talon\_get\_active\_context() -> str: Returns active context info

user.talon\_get\_hostname() -> str: Returns the hostname

user.talon\_pretty\_format(obj: object): Pretty formats an object

user.talon\_pretty\_print(obj: object): Uses pretty print to dump an object

user.talon\_sim\_phrase(phrase: Union[str, talon.grammar.vm.Phrase]): Sims the phrase in the active app and dumps to the log

user.talon\_version\_info() -> str: Returns talon & operation system verison information

user.navigation(navigation\_action: str, direction: str, navigation\_target\_name: str, before\_or\_after: str, regex: re.Pattern, occurrence\_number: int): Navigate in `direction` to the occurrence\_number-th time that `regex` occurs, then execute `navigation\_action` at the given `before\_or\_after` position.

user.navigation\_by\_name(navigation\_action: str, direction: str, before\_or\_after: str, navigation\_target\_name: str, occurrence\_number: int): Like user.navigation, but to a named target.

user.chapter\_current() -> int: Return current chapter number

user.chapter\_final(): Go to final chapter

user.chapter\_jump(number: int): Go to chapter number

user.chapter\_next(): Go to next chapter

user.chapter\_previous(): Go to previous chapter

user.debugger\_add\_hw\_breakpoint(): Add one hardware breakpoint in the debugger

user.debugger\_add\_sw\_breakpoint(): Add one software breakpoint in the debugger

user.debugger\_backtrace(): Print a back trace in the debugger

user.debugger\_break\_here(): Set a break on the current line

user.debugger\_break\_now(): Break into the debugger

user.debugger\_clear\_all\_breakpoints(): Clear all breakpoints in the debugger

user.debugger\_clear\_breakpoint(): Clear one breakpoint in the debugger

user.debugger\_clear\_breakpoint\_id(number\_small: int): Clear one breakpoint id in the debugger

user.debugger\_clear\_line(): Clear unwanted data from the command line

user.debugger\_continue(): Continue execution in the debugger

user.debugger\_detach(): Detach the debugger

user.debugger\_disable\_all\_breakpoints(): Disable all breakpoints in the debugger

user.debugger\_disable\_breakpoint(): Disable one breakpoint in the debugger

user.debugger\_disable\_breakpoint\_id(number\_small: int): Disable one breakpoint id in the debugger

user.debugger\_disassemble(): Preps the disassemble command in the debugger

user.debugger\_disassemble\_clipboard(): Disassemble instructions at an address in the clipboard

user.debugger\_disassemble\_here(): Disassembles instructions at the current instruction pointer

user.debugger\_dump\_ascii\_string(): Display as specific address as an ascii string in the debugger

user.debugger\_dump\_pointers(): Display as specific address as a list of pointers in the debugger

user.debugger\_dump\_unicode\_string(): Display as specific address as an unicode string in the debugger

user.debugger\_enable\_all\_breakpoints(): Enable all breakpoints in the debugger

user.debugger\_enable\_breakpoint(): Enable one breakpoint in the debugger

user.debugger\_enable\_breakpoint\_id(number\_small: int): Enable one breakpoint id in the debugger

user.debugger\_exit(): Exit the debugger

user.debugger\_get\_register(): Print specific register in the debugger

user.debugger\_goto\_address(): Jump to a specific address in the debugger

user.debugger\_goto\_clipboard(): Jump to a specific address stored in the clipboard

user.debugger\_goto\_highlighted(): Jump to a specific highlighted address in the debugger

user.debugger\_inspect\_type(): Inspect a specific data type in the debugger

user.debugger\_list\_modules(): List the loaded modules in the debuggee memory space

user.debugger\_restart(): Restart execution in the debugger

user.debugger\_set\_register(): Set specific register in the debugger

user.debugger\_show\_breakpoints(): Print the current breakpoints in the debugger

user.debugger\_show\_registers(): Print the current registers in the debugger

user.debugger\_start(): Start debugging

user.debugger\_step\_into(): Step into an instruction in the debugger

user.debugger\_step\_line(): Step into a source line in the debugger

user.debugger\_step\_out(): Step until function exit in the debugger

user.debugger\_step\_over(): Step over an instruction in the debugger

user.debugger\_step\_over\_line(): Step over a source line in the debugger

user.debugger\_stop(): Stop the debugger

user.file\_manager\_current\_path() -> str: Returns the current path for the active file manager.

user.file\_manager\_get\_directory\_by\_index(index: int) -> str: Returns the requested directory for the imgui display by index

user.file\_manager\_get\_file\_by\_index(index: int) -> str: Returns the requested directory for the imgui display by index

user.file\_manager\_go\_back(): file\_manager\_go\_forward\_directory

user.file\_manager\_go\_forward(): file\_manager\_go\_forward\_directory

user.file\_manager\_hide\_pickers(): Hides the pickers

user.file\_manager\_new\_folder(name: str): Creates a new folder in a gui filemanager or inserts the command to do so for terminals

user.file\_manager\_next\_file\_page(): next\_file\_page

user.file\_manager\_next\_folder\_page(): next\_folder\_page

user.file\_manager\_open\_directory(path: str): opens the directory that's already visible in the view

user.file\_manager\_open\_file(path: str): opens the file

user.file\_manager\_open\_parent(): file\_manager\_open\_parent

user.file\_manager\_open\_volume(volume: str): file\_manager\_open\_volume

user.file\_manager\_previous\_file\_page(): previous\_file\_page

user.file\_manager\_previous\_folder\_page(): previous\_folder\_page

user.file\_manager\_refresh\_title(): Refreshes the title to match current directory. this is for e.g. windows command prompt that will need to do some magic.

user.file\_manager\_select\_directory(path: str): selects the directory

user.file\_manager\_select\_file(path: str): selects the file

user.file\_manager\_show\_properties(): Shows the properties for the file

user.file\_manager\_terminal\_here(): Opens terminal at current location

user.file\_manager\_toggle\_pickers(): Shows the pickers

user.file\_manager\_update\_lists(): Forces an update of the lists (e.g., when file or folder created)

user.find(text: str): Finds text in current editor

user.find\_everywhere(text: str): Finds text across project

user.find\_next(): Navigates to the next occurrence

user.find\_previous(): Navigates to the previous occurrence

user.find\_toggle\_match\_by\_case(): Toggles find match by case sensitivity

user.find\_toggle\_match\_by\_regex(): Toggles find match by regex

user.find\_toggle\_match\_by\_word(): Toggles find match by whole words

user.replace(text: str): Search and replace for text in the active editor

user.replace\_confirm(): Confirm replace at current position

user.replace\_confirm\_all(): Confirm replace all

user.replace\_everywhere(text: str): Search and replaces for text in the entire project

user.select\_next\_occurrence(text: str): Selects the next occurrence of the text, and suppresses any find/replace dialogs.

user.select\_previous\_occurrence(text: str): Selects the previous occurrence of the text, and suppresses any find/replace dialogs.

user.camel\_left(): Moves cursor to the left by camel case/subword

user.camel\_right(): Move cursor to the right by camel case/subword

user.extend\_camel\_left(): Extends the selection by camel/subword to the left

user.extend\_camel\_right(): Extends the selection by camel/subword to the right

user.extend\_until\_line(line: int): Extends the selection from current line to the specified line

user.line\_clone(line: int): Clones specified line at current position

user.select\_range(line\_start: int, line\_end: int): Selects lines from line\_start to line line\_end

user.messaging\_channel\_next(): Move to next channel

user.messaging\_channel\_previous(): Move to previous channel

user.messaging\_mark\_channel\_read(): Mark this channel as read.

user.messaging\_mark\_workspace\_read(): Mark this workspace/server as read

user.messaging\_open\_channel\_picker(): Open channel picker

user.messaging\_open\_search(): Open message search

user.messaging\_unread\_next(): Moved to next unread channel

user.messaging\_unread\_previous(): Move to previous unread channel

user.messaging\_upload\_file(): Upload a file as a message

user.messaging\_workspace\_next(): Move to next qorkspace/server

user.messaging\_workspace\_previous(): Move to previous workspace/server

user.multi\_cursor\_add\_above(): Adds cursor to line above

user.multi\_cursor\_add\_below(): Adds cursor to line below

user.multi\_cursor\_add\_to\_line\_ends(): Adds cursor at end of every selected line

user.multi\_cursor\_disable(): Disables multi-cursor mode

user.multi\_cursor\_enable(): Enables multi-cursor mode

user.multi\_cursor\_select\_all\_occurrences(): Adds cursor at every occurrence of selection

user.multi\_cursor\_select\_fewer\_occurrences(): Removes selection & cursor at last occurrence

user.multi\_cursor\_select\_more\_occurrences(): Adds cursor at next occurrence of selection

user.multi\_cursor\_skip\_occurrence(): Skips adding a cursor at next occurrence of selection

user.page\_current() -> int: Return current page number

user.page\_final(): Go to final page

user.page\_jump(number: int): Go to page number

user.page\_next(): Go to next page

user.page\_previous(): Go to previous page

user.snippet\_create(): Triggers snippet creation

user.snippet\_hide(): Hides the snippet UI

user.snippet\_insert(text: str): Inserts a snippet

user.snippet\_search(text: str): Triggers the program's snippet search

user.snippet\_toggle(): Toggles UI for available snippets

user.split\_clear(): Clears the current split

user.split\_clear\_all(): Clears all splits

user.split\_flip(): Flips the orietation of the active split

user.split\_last(): Goes to last split

user.split\_maximize(): Maximizes the active split

user.split\_next(): Goes to next split

user.split\_number(index: int): Navigates to a the specified split

user.split\_reset(): Resets the split sizes

user.split\_window(): Splits the window

user.split\_window\_down(): Move active tab to lower split

user.split\_window\_horizontally(): Splits window horizontally

user.split\_window\_left(): Move active tab to left split

user.split\_window\_right(): Move active tab to right split

user.split\_window\_up(): Move active tab to upper split

user.split\_window\_vertically(): Splits window vertically

user.terminal\_change\_directory(path: str): Lists change directory

user.terminal\_change\_directory\_root(): Root of current drive

user.terminal\_clear\_screen(): Clear screen

user.terminal\_kill\_all(): kills the running command

user.terminal\_list\_all\_directories(): Lists all directories including hidden

user.terminal\_list\_directories(): Lists directories

user.terminal\_rerun\_search(command: str): Searches through the previously executed commands

user.terminal\_run\_last(): Repeats the last command

user.start\_repeating(key: str, ms: int): Initiate repetition

user.stop\_repeating(): Terminate repetition

user.mouse\_cardinal(direction: str, distance: int) -> int: Translate the current mouse position using a cardinal direction and a distance

user.mouse\_cardinal\_move\_1d(direction: str, distance: int): Move the mouse along a cardinal, e.g. 'move 1 left'

user.mouse\_cardinal\_move\_2d(dir1: str, dist1: int, dir2: str, dist2: str): Move the mouse along a 2d cardinal, e.g. 'move 1 left 2 up'

user.mouse\_guide\_disable(): Disable relative mouse guide

user.mouse\_guide\_enable(): Enable relative mouse guide

user.mouse\_guide\_toggle(): Toggle relative mouse guide

user.rango\_command\_with\_target(actionType: str, target: Union[str, list[str]], arg: Union[str, float, NoneType] = None): Executes a Rango command

user.rango\_command\_without\_target(actionType: str, arg: Union[str, float, NoneType] = None): Executes a Rango command without a target

user.rango\_disable\_direct\_clicking(): Disables rango direct mode

user.rango\_enable\_direct\_clicking(): Enables rango direct mode so that the user doesn't have to say 'click' before the hint letters

user.rango\_toggle\_hints(): It toggles the Rango hints globally on or off

user.rango\_type\_hotkey(): Presses the rango hotkey to read the command from the clipboard

user.win\_hide\_info() -> None: Hides the window information window

user.win\_move(direction: Optional[Dict[str, bool]] = None) -> None: Move window in small increments in the given direction, until stopped

user.win\_move\_absolute(x: float, y: float, region: Optional[Dict[str, bool]] = None) -> None: Move window to given absolute position, centered on the point indicated by the given region

user.win\_move\_percent(percent: float, direction: Optional[Dict[str, bool]] = None) -> None: Move window some percentage of the current size

user.win\_move\_pixels(distance: int, direction: Optional[Dict[str, bool]] = None) -> None: Move window some number of pixels

user.win\_move\_to\_pointer(region: Optional[Dict[str, bool]] = {'horizontal': False, 'vertical': False, 'diagonal': False}): Move window to pointer position, centered on the point indicated by the given region

user.win\_resize\_absolute(target\_width: float, target\_height: float, region: Optional[Dict[str, bool]] = None) -> None: Size window to given absolute dimensions, optionally by stretching/shrinking in the direction indicated by the given region

user.win\_resize\_percent(percent: float, direction: Optional[Dict[str, bool]] = None) -> None: Change window size by a percentage of current size

user.win\_resize\_pixels(distance: int, direction: Optional[Dict[str, bool]] = None) -> None: Change window size by pixels

user.win\_resize\_to\_pointer(nd\_direction: Dict[str, bool]) -> None: Stretch or shrink window to pointer position, centered on the point indicated by the given region

user.win\_revert() -> None: Restore current window's last remembered size and position

user.win\_show\_info() -> None: Shows information about current window position and size

user.win\_shrink(direction: Optional[Dict[str, bool]] = None) -> None: Shrink window in small increments until stopped, optionally in the given direction

user.win\_snap\_percent(percent: int) -> None: Center window and change size to given percentage of parent screen (in each direction)

user.win\_stop() -> None: Module action declaration for stopping current window move/resize operation

user.win\_stretch(direction: Optional[Dict[str, bool]] = None) -> None: Stretch window in small increments until stopped, optionally in the given direction

user.win\_test\_bresenham(num: int) -> None: Test modified bresenham algo

user.center\_eagle(): move mouse to center of screen

user.display\_mode(mode: str): change how much information is displayed in the compass grid

user.eagle\_disable(): Disable relative mouse guide

user.eagle\_enable(): Enable relative mouse guide

user.eagle\_head\_start(bearing: float): enable relative mouse guide and point to given bearing direction

user.eagle\_toggle(): Toggle relative mouse guide

user.fly\_back(distance: int): turn around and move back the specified number of pixels

user.fly\_out(distance: int): move out the specified number of pixels

user.move\_cardinal(move\_degrees: int, target: float): move the bearing direction a certain number of degrees towards a cardinal direction

user.reverse(): reverse direction

user.set\_cardinal(target: float): set the bearing to a cardinal direction

user.test(d1: float): test function

user.mouse\_helper\_blob\_picker(bounding\_rectangle: talon.types.rect.Rect, min\_gap\_size: int = 5): Attempts to find clickable elements within the given bounding rectangle, then: draws a labelled overlay allowing you to click or move the mouse to them.

: See mouse\_helper\_calculate\_relative\_rect for how to get the bounding rectangle.

user.mouse\_helper\_calculate\_relative\_rect(relative\_rect\_offsets: str, region: str = 'active\_screen') -> talon.types.rect.Rect: Calculates a talon rectangle relative to the entire screen based on the given region: of interest and a set of offsets. Examples:

: "0 0 -0 -0", "active\_screen": Would indicate the entire active screen.: "10 20 30 40", "active\_window": Would indicate the region between pixels (10, 20) and (30, 40): on the currently focussed window.: "10 20 -30 40", "active\_window": Would indicate the region between pixels (10, 20) and: the pixel 30 units from the right hand side of the window and 40 units from the top.

user.mouse\_helper\_find\_template\_relative(template\_path: str, xoffset: float = 0, yoffset: float = 0, region: Optional[talon.types.rect.Rect] = None) -> List[talon.types.rect.Rect]: Finds all matches for the given image template within the given region.

: :param template\_path: Filename of the image to find. Can be an absolute path or: if no '/' or '\' character is specified, it is relative to the image: templates directory.: :param xoffset: Amount to shift in the x direction relative to the: center of the template.: :param yoffset: Amount to shift in the y direction relative to the: center of the template.: :param region: The region to search for the template in. Either a screen relative: TalonRect (see mouse\_helper\_calculate\_relative\_rect) or None to just use the: active screen.

user.mouse\_helper\_move\_active\_window\_relative(xpos: str, ypos: str): Positions the mouse relative to the active window

user.mouse\_helper\_move\_image\_relative(template\_path: str, disambiguator: Union[int, str] = 0, xoffset: float = 0, yoffset: float = 0, region: Optional[talon.types.rect.Rect] = None): Moves the mouse relative to the template image given in template\_path.

: :param template\_path: Filename of the image to find. Can be an absolute path or: if no '/' or '\' character is specified, it is relative to the image: templates directory.: :param disambiguator: If there are multiple matches, use this to indicate: which one you want to match. Matches are ordered left to right top to: bottom. If disambiguator is an integer then it's just an index into that list.: If it's the string "mouse" then it's the next match in the region to the right: and down from the mouse after shifting back the offset amount and up and left: half the size and width of the template. If it is "mouse\_cycle" then if there: are no further matches it will attempt to start from the top of the screen again.: This is useful for iterating through rows in a table for example.: :param xoffset: Amount to shift in the x direction relative to the: center of the template.: :param yoffset: Amount to shift in the y direction relative to the: center of the template.: :param region: The region to search for the template in. Either a screen relative: TalonRect (see mouse\_helper\_calculate\_relative\_rect) or None to just use the: active screen.

user.mouse\_helper\_move\_relative(xdelta: float, ydelta: float): Moves the mouse relative to its current position

user.mouse\_helper\_position\_restore(): Restores a saved mouse position

user.mouse\_helper\_position\_save(): Saves the mouse position to a global variable

user.command\_wizard\_choose\_option(option: int): Chooses one of the command wizards

user.command\_wizard\_hide(): Closes the command wizard UI

user.command\_wizard\_show(): Brings up the command wizard UI

user.marker\_ui\_hide(): Hides any visible marker UI

user.marker\_ui\_mouse\_move(label: str): Moves the mouse cursor to the label corresponding to the given label

user.marker\_ui\_show(rects: List[talon.types.rect.Rect]): Shows the given markers in the Marker UI. They can then be clicked or moved: to using other actions in this class.

user.formatted\_text(phrase: Union[str, talon.grammar.vm.Phrase], formatters: str) -> str: Formats a phrase according to formatters. formatters is a comma-separated string of formatters (e.g. 'CAPITALIZE\_ALL\_WORDS,DOUBLE\_QUOTED\_STRING')

user.formatters\_reformat\_last(formatters: str) -> str: Clears and reformats last formatted phrase

user.formatters\_reformat\_selection(formatters: str) -> str: Reformats the current selection.

user.get\_formatters\_words() -> dict: returns a list of words currently used as formatters, and a demonstration string using those formatters

user.insert\_formatted(phrase: Union[str, talon.grammar.vm.Phrase], formatters: str): Inserts a phrase formatted according to formatters. Formatters is a comma separated list of formatters (e.g. 'CAPITALIZE\_ALL\_WORDS,DOUBLE\_QUOTED\_STRING')

user.insert\_many(strings: list[str]) -> None: Insert a list of strings, sequentially.

user.insert\_with\_history(text: str): Inserts some text, remembering it in the phrase history.

user.reformat\_text(text: str, formatters: str) -> str: Reformat the text.

user.move\_cursor(s: str): Given a sequence of directions, eg. 'left left up', moves the cursor accordingly using edit.{left,right,up,down}.

user.create\_spoken\_forms(source: str, words\_to\_exclude: Optional[list[str]] = None, minimum\_term\_length: int = 3, generate\_subsequences: bool = True) -> list[str]: Create spoken forms for a given source

user.create\_spoken\_forms\_from\_list(sources: list[str], words\_to\_exclude: Optional[list[str]] = None, minimum\_term\_length: int = 3, generate\_subsequences: bool = True) -> dict[str, str]: Create spoken forms for all sources in a list, doing conflict resolution

user.create\_spoken\_forms\_from\_map(sources: Mapping[str, Any], words\_to\_exclude: Optional[list[str]] = None, minimum\_term\_length: int = 3, generate\_subsequences: bool = True) -> dict[str, typing.Any]: Create spoken forms for all sources in a map, doing conflict resolution

user.add\_selection\_to\_vocabulary(phrase: Union[talon.grammar.vm.Phrase, str] = '', type: str = ''): Permanently adds the currently selected text to the vocabulary with the provided: spoken form and adds variants based on the type ("noun" or "name").

user.add\_selection\_to\_words\_to\_replace(phrase: talon.grammar.vm.Phrase, type: str = ''): Permanently adds the currently selected text as replacement for the provided: original form and adds variants based on the type ("noun" or "name").