openstenoproject / plov…

Sign up

Code Issues 123 Pull requests 6 Projects 0 Wiki Pulse

Document your code Dismiss

Every project on GitHub comes with a version-controlled wiki to give your documentation the high level of care it deserves. It's easy to create well-maintained, Markdown or rich text documentation alongside your code.

Sign up for free

See pricing for teams and enterprises

Dictionary Format

SeaLiteral edited this page Feb 18, 2019 · 64 revisions

Jump to bottom

This page contains an explanation for the syntax used in Plover's dictionaries, as well as the operators that control capitalization, spacing, keyboard shortcuts, and other commands.

Table of Contents

Strokes and Dictionaries

If you are new to stenography:

- A *stroke* is a group of keyboard keys that are pressed simultaneously to produce output. As soon as all keys on the keyboard are released, the steno machine sends out that entire chord of key presses as a stroke.
- Plover converts that input into English words and phrases on the screen, by using *dictionaries*.
- The dictionaries map a stroke (or a combination of strokes) to an English word or phrase, number, punctuation symbol, emoji, software command, or a meta command for controlling spacing and capitalization.
- Every stenographer creates their own customized dictionary to supplement or amend the default dictionary that comes with a steno application, such as Plover. By default, you add your customizations to the `user.json` dictionary that comes with Plover.

Plover's Default Dictionaries

Plover comes supplied with three dictionaries:

- `main.json`. This contains the default dictionary. It is based on Mirabai Knight's own personal dictionary, which follows a StenEd-style thoery. It contains over 140,000 entries and is adequate for anyone learning stenography. Mirabai uses this dictionary professionally for her realtime work.
- `commands.json`. This contains Plover Control Commands.
- `user.json`. This is available for your personal customizations. `user.json` is a blank dictionary. By default, the `user.json` dictionary is at the bottom of the dictionary list and has the highest priority - it is the first dictionary Plover will use. When you define new strokes, they will get added to this dictionary. This means you can see which strokes you've defined yourself, instead of trying to locate them inside the default dictionaries.

You can add extra dictionaries to Plover, as well. Dictionaries may be located on your system in any directory/folder; they do not have to be in a subdirectory/subfolder of the Plover installation.

> **Note:** We do not recommend you remove the `commands.json` dictionary from the dictionary list. This is because Plover has some concepts that users of other steno software will not be familiar with initially.

Dictionary priority

If two dictionaries contain the same steno strokes, Plover will use the one in the dictionary that has the highest priority. The dictionaries in the dictionary list are prioritised from the bottom up.

By default, the `user.json` dictionary is placed at the bottom of the dictionary list and has the highest priority. If you want new strokes to go to different dictionary (for example, you have your own dictionary already), make sure that dictionary is at the bottom of the list.

> **Note:** With Plover 4.0+, default dictionary priority is now top-down and can be changed. In the configuration settings, you can select your preferred ordering of dictionary priority to be bottom-up or top-down.

JSON and RTF/CRE

Plover supports two types of dictionaries:

- **JSON** (the default and recommended format), and
- **RTF/CRE**.

RTF/CRE is an import/export format used by proprietary steno software. This means Plover can work with exported dictionaries from Eclipse, ProCAT, Case CATalyst, and other steno software applications.

Limitations

There are some limitations with each format:

- RTF dictionaries will cause Plover to take longer to start up and won't have Unicode support.
- The JSON format that Plover uses is not used or supported by other steno applications. If you want to move or copy your Plover JSON formatted dictionary to another steno software application, you will need to convert it to RTF.
- The Plover JSON format doesn't have support for stroke metadata. At the moment, Plover doesn't support reading/writing of RTF metadata.

Plover Control Commands

You can control some aspects of Plover withstrokes. Plover's default dictionary (`commands.json`) contains these commands:

Command name	Command	Default Stroke	Description
Add Translation	`{PLOVER:ADD_TRANSLATION}`	`TKUPT` (think DUPT for "Dictionary UPdaTe")	Opens a translation window where you can enter a stroke and translation text to create a new dictionary entry.
Disable Output	`{PLOVER:SUSPEND}`	`PHRO*F` (think PLOF for "PLover OFf")	Stop translating steno. With a keyboard machine, you will be able to type normally again.
Enable Output	`{PLOVER:RESUME}`	`PHROPB` (think PLON for "PLover ON")	Re-enable Plover's output. You can write this stroke to switch back from your keyboard into steno mode.
Toggle Output	`{PLOVER:TOGGLE}`	`PHROLG` (think PLOLG, for PLOver toGGLe)	Toggle between output being enabled and disabled.

Command name	Command	*Suggested* Stroke	Description
Look Up Stroke	`{PLOVER:LOOKUP}`	`PHR*UP`	Open a search dialog that you write a translation into to get a list of entries in your dictionaries.
Configure	`{PLOVER:CONFIGURE}`	`PHROFG` (think PLOFG, for PLOver conFiG)	Open and focus the Plover configuration window.
Focus	`{PLOVER:FOCUS}`	`PHROFBGS` (think PLOFKS, for PLOver focus)	Open and focus the main Plover window.
Quit	`{PLOVER:QUIT}`	`PHROBGT` (think PLOKT, for PLOver **quit**)	Quit Plover entirely.

Sending Symbols

Since Plover 3.0.0, users can write full Unicode. You can use symbols in your JSON dictionary, or paste them into the "Add Translation" dialog (or into the dictionary manager). You can get symbols from a character insert panel in your operating system (if you have one), or you can copy and paste symbols from another application, such as a web browser.

```
{
  "SHR*UG": "¯\\_(□)_/¯",
  "TR*PL": "{^}™",
  "TPHA*ES": "n—o— yes"
}
```

Text Formatting

Prefixes, Infixes, and Suffixes

Strokes can be attached at the beginning and/or end using the "attach" operator. You can also use some of the built-in orthographic rules that Plover uses for creating intelligent strokes.

- `{^}` is the attach operator.
- `{^ish}` is an orthographic-aware suffix that will add "ish" to the end of the previous word. E.g. `RED/EURB` will output `reddish`. Note: addition of a second "d" caused by Plover's understanding of English orthography.
- `{^}ish` is a suffix with the text outside the operator—this means that the text will simply be attached without grammar rules. Using this stroke in the previous example would give instead `redish`.
- `{^-to-^}` is an infix, e.g. `day-to-day`.
- `{in^}` is a prefix, e.g. `influx`.
- Most custom punctuation entries will take advantage of the attach operator, e.g. `{^—^}` for an emdash.

Glue Operator (Numbers, Fingerspelling)

Glue is sort of like the attach operator above, except that "glued" strokes only attach to neighboring "glue" strokes.

Translations containing *only* digits are glued, allowing you to output multiple number strokes to make a large number.

- `{&}` is the glue operator.
- `{&a}`, `{&b}`, `{&c}`, etc. are how the fingerspelling alphabet is made.
- `{&th}` is a multiletter glue stroke, which can be useful (`TH*` in the default dictionary)

Because glued translations only glue to other glued translations, they won't attach to regular words. This gives us some power because you can write:

`THR/-R/#H/#A/KATS` to get "there are 45 cats" and only `#H` (4) and `#A` (5) are "glued" to each other.

Capitalizing

Capitalizing a word means turning the first letter into a capital, e.g. proper nouns and in titles. Usually your dictionary has capitals pre-defined, but there are times when you need to capitalize a word on the go.

Capitalize Next Word

- `{-|}`

The next word will have a capitalized first letter. In the default dictionary, we have `"KPA": "{-|}"`, which will capitalize the next word; and `"KPA*": "{^}{-|}"` which omits a space and capitalizes the next word. That last stroke would let you writeLikeThis. This formatting style is called "camel case', which some programmers use in their work.

Capitalize next word can also be used in name titles, like `Ms. {-|}`.

Default strokes:

- `KPA`: `{-|}` (think "cap")

- `KPA*` : `{^}{-|}` (also suppresses space)

Capitalize Last Word

- `{*-|}`

The last stroke word will have a capitalized first letter. This is useful in case you realize that a word should be capitalized after you've written it. It can also be used in a stroke, like in `{*-|}{^ville}`. This would capitalize the last word and attach to it. This would be useful for town names on the fly, such as `Catville`.

Suggested stroke: `KA*PD`

Carrying Capitalization

- `{~|text}` or `{^~|text^}` where the attach operator is optional and the text can be changed.

In English, we have punctuation that doesn't get capitalized, but instead the next letter gets the capitalization. For example, if you end a sentence in quotes, the next sentence still starts with a capital letter! `"You can't eat that!" The baby ate on.`

In order to support this, there is a special pre/in/suffix syntax that will "pass on" or "carry" the capitalized state. You might find this useful with quotes, parentheses, and words like `'til` or `'cause`.

The default dictionary for Plover should use these operators where appropriate.

```
{
  "KW-GS": "{~|\"^}",
  "KR-GS": "{^~|\"}",
  "KA*US": "{~|'^}cause",
  "PREPB": "{~|(^}",
  "PR*EPB": "{^~|)}"
}
```

For a newline, the syntax would be `{^~|\n^}`.

Uppercasing (CAPS)

See Output Modes for CAPS mode, which acts like CAPS lock on a regular keyboard. Alternatively, you can use a Keyboard Shortcut set to `{#Caps_Lock}` to activate the system CAPS lock like you can on your keyboard.

Suggested stroke: `"KA*PS": "{MODE:CAPS}"`

Uppercase Next Word

- `{<}`

Output next stroke in capital letters, e.g. `{<}cat` → `CAT`

Suggested stroke: `KPA*L` (cap all)

Uppercase Last Word

- `{*<}`

Rewrite last word in capital letters, e.g. `cat{*<}` → `CAT`

Suggested stroke: `*UPD`

Lowercasing

Lowercase Next Word

- `{>}`

Forces the next letter to be lowercase, e.g. `{>}Plover` → `plover`

Suggested stroke: `HRO*ER` (lower)

Lowercase Last Word

- `{*>}`

Rewrite the last word to start with a lowercase letter, e.g. `Plover{*>}` → `plover`

Suggested stroke: `HRO*ERD` (lowered)

Canceling Formatting of Next Word

In order to cancel formatting of the next word, use the empty meta tag as your definition:

- `{}`

Using `{}` in front of a arrow key commands, as in `{}{#Left}`, is useful if the arrow key commands are used to move cursor to edit text. Canceling formatting actions for cursor movement prevents Plover from, for instance, capitalizing words in middle of a sentence if cursor is moved back when the last stroke, such as `{.}`, includes action to capitalize next word.

See also the "do nothing" translation

Format Currency

There is a built-in meta in Plover that allows you to format the last-written number as currency. The format is `{*($c)}` where `$` is any currency symbol you'd like, and `c` is the amount, formatted in standard currency format, with either no decimals or two. Commas are added every 3 numbers before the decimal automatically.

- `{*($c)}`: Standard English dollars
  - `23{*($c)}` → $23
  - `2000.5{*($c)}` → $2,000.50
- `{*($c CAD)}`: You can include other text, e.g. when specifying a currency's country
  - `100{*($c CAD)}` → $100 CAD
- `{*(c )}`: The symbol can be placed on either side of the number, which often happens in languages other than English and in certain regions.
  - `2345{*(c )}`: 2,345

Here are some other currency symbols, in case you need to copy-paste them into your entries: £, ¥, €, $, ₩, ¢

Undoable Line Breaks and Tabs

When you use keyboard shortcuts, the asterisk/undo command on Plover will not have any effect. This is a limitation imposed by the fact that most commands will not have a meaningful undo. For example, you wouldn't "undo" a "copy" command. For this reason, `{#return}` and `{#tab}` don't work how many users would expect.

Instead, we must use special characters that can be undone by Plover for new paragraphs and erasable tabs. Specifically:

- `\n` or `\r` for line breaks.
- `\t` for tabs.

For example:

- `{^\n^}{-|}`

  This translation would create a line break without spacing and then capitalize the next word. It can be removed with the asterisk key.

- `{^\t^}`

  This translation presses the tab key without any other spacing. It can be undone with the asterisk key.

Macros

Macros can be mapped from a stroke and perform operations on the stroke-level. This means that it can perform actions based on previous strokes, not necessarily on previous words or translations.

Undo / Delete Last Stroke

- `=undo`

The built-in "undo" macro is assigned to the asterisk key `*`. You can map other strokes to "undo" or "delete last stroke" by creating a stroke where the translation is `=undo`

Repeat Last Stroke

- `{*+}`

  A stroke mapping to this command will send the last stroke entered. A common stroke to map to repeat-last is the bare number bar; `"#": "{*+}"`; causing `KAT/#/#` to behave like `KAT/KAT/KAT`.

  Repeat last stroke `{*+}` is very useful for keys that you repeat. For example, when you are moving around text in a document.

Suggested stroke: `#`

Toggle asterisk

- `{*}`

  A stroke mapping to this command will toggle the asterisk key on the last stroke entered. For example, if you had this stroke in your dictionary as Number Bar + Asterisk, `"#*": "{*}"`, when you write `KAT/#*` it will behave as if you wrote `KA*T`.

  Toggle asterisk `{*}` is useful for when you notice that you should have included an asterisk in your last stroke. For example, you wanted to write the name "Mark" (a person's name), but you wrote "mark" (the noun/verb). At this point, you can use Toggle asterisk `{*}` to correct it. You wouldn't have to erase the word and re-stroke it (this time, with you including the missing the asterisk).

Suggested stroke: `#*`

Retrospectively Add Space

- `{*?}`

  A stroke mapping to this command will add a space between your last stroke and the one before that, splitting apart the two strokes. For example, if your dictionary contained `PER` as "Perfect", `SWAEUGS` as "Situation" and `PER/SWAEUGS` as "Persuasion". If you meant to write "Perfect situation", but saw your output was "Persuasion", you could force these two strokes to be split apart by using the `{*?}` stroke. This means your output would change on the screen from "Persuasion" to "Perfect situation".

Suggested stroke: `AFPS` (add space)

Retrospectively Delete Space

- `{*!}`

  A stroke mapping to this command will delete the space between your last stroke and the one before that. If you wrote "Basket ball" but wanted it to be "Basketball", you could force these strokes together by using the {*!} stroke. Plover will go back and remove the space before your last stroke; As a result, your output would become "Basketball".

Suggested stroke: `TK-FPS` (**d**elete space)

Keyboard Shortcuts

Most Plover strokes are just text and formatting operators. Plover handles standard strokes really well. This allows it to handle "undoing" with the asterisk key, as well as automatically handling case and spacing. However, Plover's text and formatting strokes can't send arbitrary key strokes, such as sending keyboard shortcuts.

Note: Plover 3.1 introduces new rules for commands that differ slightly from the previous implementation. Before Plover 3.1, commands were used to send symbols because Plover didn't have full Unicode support. *It used to be possible to send "+" by writing* `{#plus}`*, but the system has been updated.*

- `{#}` is the command operator.

Inside of a command block, you write in what keys you want Plover to simulate. The keys hit in command blocks won't be "undone" when using the asterisk (because you can't backspace a keyboard shortcut). You select the keys by their name. All key names will only refer to the base key. For example, to use letter keys, you just use the corresponding letter (case insensitive) separated by spaces:

- `{#a b c d}` will send "abcd" and will not affect Plover's text formatting or asterisk undo-buffer.

You can also use key names, which is needed when you are accessing a symbol key. For example, on the QWERTY layout there is an equal/plus key in the top right, which you can access by either of its names:

- `{#equal plus}` will send "==". Plover doesn't send modifiers. It just hits the key based on the name.

If you want to send symbols, though, don't use commands. Commands should be used for keyboard shortcuts only. Instead, use the symbol in your dictionary entry.

Modifier Names

If you want to use a modifier, use it by name (e.g. `Shift_L`). For convenience, all key names are case insensitive and you can optionally default to the left modifier by dropping the side selector:

Modifier	Command Key Names (case-insensitive)
Shift	`Shift_L`, `Shift_R`, `shift`
Control	`Control_L`, `Control_R`, `control`
Alt	`Alt_L`, `Alt_R`, `alt`, `option`
Super	`Super_L`, `Super_R`, `super`, `windows`, `command`

For modifiers, use parentheses to delimit where the keys are pressed down.

Shortcut Key Names

Here are the key names you'll want to use:

Keys	Command Key Names (case-insensitive)
Letters	`a`, `b`, …, `z`
Accented Letters (international layouts)	`udiaeresis`, `eacute`, etc.
Numbers	`0`, `1`, …, `9`
Control Keys	`Escape`, `Tab`, `Caps_Lock`, `space`, `BackSpace`, `Delete`, `Return`, etc.
F-Keys	`F1`, `F2`, …, `F12`
Common Named Keys	`asciitilde` (~), `asciicircum` (^), `equal`, `minus`, `slash`, `backslash`, `comma`, `colon`, etc.
Media Keys	**Common**: `AudioRaiseVolume`, `AudioLowerVolume`, `AudioMute`, `AudioNext`, `AudioPrev`, `AudioStop`, `AudioPlay`, `AudioPause`, `Eject`, **Mac**: `MonBrightnessUp`, `MonBrightnessDown`, `KbdBrightnessUp`, `KbdBrightnessDown`, **Windows**: `Back`, `Forward`, `Refresh`. Note: Linux supports supports any XF86 keyname.

Consult the code for the full list of supported keyboard shortcut keys.

Note that a particular key name will send an unmodified key. For example, for many typical keyboard layouts `{#braceleft}` will cause `[` to be outputted while `{#shift(braceleft)}` will cause `{` to be outputted.

Most symbols (e.g. `+, =, ~, r`) can just be included directly in the definition. But some symbols are part of the dictionary syntax and so need to be escaped:

Symbol	Escaped Form
`"`	`\"`
`\`	`\\`
`{`	`\\{`
`}`	`\\}`

Example Shortcuts

Here are some shortcuts. They are in JSON format:

- `"STPH-G": "{#right}"` — right arrow on the keyboard, for moving the cursor to the right once
- `"SKWR-G": "{#shift(right)}"` — shift and right arrow on the keyboard, for selecting one character
- `"SKWR-BG": "{#control(shift(right))}"` — shift, control, and right arrow on the keyboard, for selecting one word to the right on Windows/Linux
- `"SKWR-BG": "{#option(shift(right))}"` — option (alt), control, and right arrow on the keyboard, for selecting one word to the right on Mac

These next strokes are not particularly useful, but they show you what you can do with the command syntax:

- `"TKAO*UP": "{#control(c v v v)}"` — copy, then paste 3 times
- `"SKPH-Z": "{#control(z shift(z))"` — program-dependent, but possibly "undo/redo". Notice how the first z has only the control operator, and the second has both the control and the shift operator.

Commands are case insensitive - adding capitals will not affect the output. `{#control(z shift(z))` is the same as `"{#CONTROL_L(Z SHIFT(Z))}"`

"Do Nothing" Translation

You can use the keyboard shortcut syntax (`{#}`) if you want a stroke that effectively does nothing. For example: a null or canceled stroke, which doesn't affect formatting, doesn't output anything, and cannot be "undone" with the asterisk key. A stroke mapped to `{#}` will effectively do nothing but show up in your logs.

- `{#}` an effective "null" stroke.

See also: Canceling Formatting of Next Word

Output Modes

- `{MODE:}` is the mode operator

Plover supports special character casing, such as CAPS LOCK and Title Case. It also supports replacing its implicit space with other characters. This is useful for when you want to write_in_snake_case.

Output modes are activated with a special syntax in your dictionary. They can be a stroke or just part of a stroke. They can then be turned off with a special reset command.

An example flow for CAPS LOCK might be:

1. Turn on CAPS LOCK.
2. Write in capital letters.
3. Turn off CAPS LOCK.

However, there's nothing stopping you from building in output modes into your strokes. For example, you might want your new paragraph stroke to reset the case mode, no matter what.

Reset Command

You can reset the output mode to its default with `{MODE:RESET}`.

> **Important**: We recommended you have a reset output mode command, so that you can always reset Plover's output mode. This is in case you change it by accident.

- `{MODE:RESET}`: Reset the case and space character. We recommended this for getting out of any custom output mode. For example: `"R-R": "{^~|\n^}{MODE:RESET}"` and `"TPEFBG": "{#escape}{MODE:RESET}"`
- `{MODE:RESET_CASE}`: Exit caps, lower, or title case.
- `{MODE:RESET_SPACE}`: Use spaces as normal.

Modes

There are some built-in modes you can use:

Dictionary Syntax	Sample Output
`{MODE:CAPS}`	THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG.
`{MODE:TITLE}`	The Quick Brown Fox Jumps Over The Lazy Dog.
`{MODE:LOWER}`	the quick brown fox jumps over the lazy dog.
`{MODE:CAMEL}`	theQuickBrownFoxJumpsOverTheLazyDog.
`{MODE:SNAKE}`	The_quick_brown_fox_jumps_over_the_lazy_dog.

Custom Modes

You can define your own custom modes with the `SET_SPACE:` operator. This allows you to replace the space that Plover outputs with anything. Plover's snake-mode is the same as `SET_SPACE:_`.

Here are some other examples:

Dictionary Syntax	Sample Output
`{MODE:SET_SPACE:}`	Thequickbrownfoxjumpsoverthelazydog.
`{MODE:SET_SPACE:-}`	The-quick-brown-fox-jumps-over-the-lazy-dog.
`{MODE:SET_SPACE:😁}`	The😁quick😁brown😁fox😁jumps😁over😁the😁lazy😁dog.

Summary of suggested commands you can cut and paste into your dictionary

Here is a summary of the suggested commands you can cut and paste into your personal dictionary:

```
{
"PHR*UP":"{PLOVER:LOOKUP}",
"PHROFG":"{PLOVER:CONFIGURE}",
"PHROFBGS":"{PLOVER:FOCUS}",
"PHROBGT":"{PLOVER:QUIT}",
"KA*PD":"{*-|}",
"KA*PS":"{MODE:CAPS}",
"KPA*L":"{<}",
"*UPD":"{*<}",
"HRO*ER":"{>}",
"HRO*ERD":"{*>}",
"#":"{*+}",
"#*":"{*}",
"AFPS":"{*?}",
"TK-FPS":"{*!}"
}
```

> **Note:** The final entry must not have a trailing comma.

Plover Wiki — Open Steno Project — Plover Homepage

▸ **Pages** 27

Plover

- Homepage (external)
- Installation Guide
- Supported Hardware
- Get Started with Plover
- Dictionary Format
- Built-in Tools
- Open Steno Project Timeline
- Troubleshooting Issues

Stenography

- Stenography Overview
- Plover and Professional Stenography
- Glossary
- Learning Resources
- Brief Ideas
- List of Available Steno Dictionaries

The Steno Community

In-Browser Demo

Other Pages

Clone this wiki locally

```
https://github.com/openstenoproject/plover.wiki.git
```

© 2019 GitHub, Inc.