↖ Topic Index

Emacs support for the Go Programming Language 🚧

Description	Keystroke	Function	Note

Go Support
○ Help & Customization
• Info
• Navigation
○ Indentation
• Syntax Check
	Support for the **Go programming language** is described in this page.
	📦 Go support requires the **go-mode** external package, which supports the classic **go-mode** and tree-sitter based **go-ts-mode**.
	PEL supports it when **pel-use-go** user options are turned on (either t for using the classic major mode, or with-tree-sitter to use **go-ts-mode**.
	
	Tree Sitter support for Go under PEL:
	• **Requirements**: Use **Emacs >= 30.** Emacs must be built with tree-sitter support. **pel-use-tree-sitter must nee** set to t. Go Tree Sitter grammar installed.
	• Activate tree-sitter for Go by setting **pel-use-go** to 'with-tree-sitter to use **go-ts-mode**. If **pel-use-go** is set to **t**, Emacs will use the classic **go-mode.**
	• During the Emacs session you can change the value of **pel-use-go** to change the major mode used.
	• If the gomod tree sitter grammar is installed, and **pel-use-go** is set to 'with-tree-sitter, the go.mod files are opened with the **go-mod-ts-mode**, otherwise they are opened with the **go-dot-mod-mode** provided by **go-mode**.
	
	PEL also provides:
	• **Speedbar** support for .go files listing functions and types. **iMenu** is supported by **go-mode** and **go-ts-mode**.
	• A Go section is added to **Menus** by **go-mode** but nothing by **go-ts-mode**.
	• Automatic execution of **gofmt** when saving a buffer into a file when **pel-use-gofmt-on-buffer-save** user option is set to t.
	• Generic programming language features like template text insertion handle Go comment style. See **Inserting Text** .
	• Control of the tab width for all go files, via the **pel-go-tab-width** user-option (access the PEL customer buffer with the **<f11> SCP g <f2>** key sequence or **<f12> <f2>** from inside a buffer visiting a Go source code file or via the button for it in the buffer opened by pel-go-setup-info (bound to **<f12> ?**).
	  • The value of **tab-width** (used by **go-mode**) and the value of **go-ts-mode-indent-offset** (used by **go-ts-mode**) are set to the value of **pel-go-tab-width** when a Go buffer is opened, ensuring consistency if you switch from one major mode to the other.
	  ☝ The Go language officially uses hard tabs for indentation and the gofmt program replaces spaces used for indentation by hard tabs. Although unusual for several, it's actually quite nice policy along as you use the same value for a hard tab visual width and the indentation width (as PEL does).
	  ☝ With that in place you can change the visual rendering of indentation by changing these values. With PEL, simply change the **pel-go-tab-width**, make sure the buffer is saved and refresh it with revert-buffer (bound to **<f11> f r f)** . New buffers will use the new visual rendering of the indentation.
	• Support for syntax checking with either flymake or **flycheck** via the **goflymake** Go program when **pel-use-goflymake** user option is set to t.		

Last updated on:	2025-10-13		All support requires support for the Go programming language installed on your computer. • See **Go installation instructions** or use Homebrew's command **brew install go**.
Open this PDF file. See also: **Help/Info**	**<f11> SPC g <f1>** **<f12> <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **Go** local PDF. If the prefix argument (like **C-u** or **M--**) is used, then it opens the remote GitHub hosted raw PDF instead. If the **pel-flip-help-pdf-arg** user-option is set it's the other way around.
Customize PEL Go support	**<f11> SPC g <f2>** **<f12> <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL Go support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Customize Emacs Go support	**<f11> SPC g <f3>** **<f12> <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Go support: go, go-cover, godoc, go-dot-mod. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Show PEL setup for Go	**<f11> SPC g ?** **<f12> ?**	(**pel-go-setup-info** &optional APPEND)	Display Go setup information inside a *pel-go-info* buffer with buttons providing quick access to the customization buffer of each variable shown. The information shown includes the value and interpretation of: • pel-use-go (whether the classic or tree-sitter based major mode is used). • pel-go-tab-width, tab-width and go-ts-mode-indent-offset • pel-use-gofmt-on-buffer-save: whether gofmt is executed before saving buffer. • pel-use-goflymake To append information in the buffer instead of clearing the previous content type any prefix argument (such as **C-u**) before the command keystroke.
Describe expression at point.	**C-c C-d**	(**godef-describe** POINT)	Describe the expression at POINT. 💻 This uses the **godef executable**, a Go program. • To install it, run the following command from a shell: `go get github.com/rogpeppe/godef`. • The GOPATH environment variable must be setup and GOPATH/bin must be in the PATH to be able to run godef.
Set visual rendering of hard tabs for the current buffer	**<f11> M-t**	(**pel-set-tab-width** N)	Change the tab width of the current buffer, only affecting the display rendering of hard tabs inserted in the buffer text. Prompts for a new value in the [2, 8] range. • This modifies a buffer local value of the the **tab-width** user-option. • The change is temporary and affects the current buffer only. • To change the tab width used for all Go source code files, change the '**pel-go-tab-width**' user-option variable instead. See **Indentation** for more information.
Toggle gofmt run on buffer save	**<f11> SPC g M-s** **<f12> M-s**	(**pel-go-toggle-gofmt-on-buffer-save** &optional GLOBALLY)	Toggle automatic run of **gofmt** when saving Go buffer to file. • By default change behaviour for local buffer only. • When GLOBALLY argument is non-nil, change it for all Go buffers for the current Emacs editing session (the change does not persist across Emacs sessions). • To modify the global state permanently modify the customized value of the **pel-go-toggle-gofmt-on-buffer-save** user option via the 'pel-pkg-for-go'group customize buffer.
Inserting code	See also: **Inserting Text** generic commands that apply to go buffers.		
Add new import package to list of module package import statement	**C-c C-a**	(**go-import-add** ARG IMPORT)	Add a new IMPORT to the list of imports. Don't move point. • When called with a prefix ARG asks for an alternative name to import the package as. • If no list exists yet, one will be created if possible. • If an identical import has been commented, it will be uncommented, otherwise a new import will be added.
Navigation	See also: **Navigation** generic commands that apply to go buffers. The main commands are shown here but more are available and described there.		
Move to expression definition	**C-c C-j**	(**godef-jump** POINT &optional OTHER-WINDOW)	Jump to the definition of the expression at POINT. • after that command, use **M-,** to go back to original point.
Move to expression definition in other window	**C-x 4 C-c C-j**	(**godef-jump-other-window** POINT)	Jump to the definition of the expression at POINT but into the other window. • after that command, use **M-,** to go back to original point.
Move to current function arguments	**C-c C-f a**	(**go-goto-arguments** &optional ARG)	Go to the arguments of the current function. • If ARG is non-nil, anonymous functions are skipped.
Move to current function docstring	**C-c C-f d**	(**go-goto-docstring** &optional ARG)	Go to the top of the docstring of the current function. • If there is none, add one beginning with the name of the current function. • Anonymous functions do not have docstrings, so when this is called interactively anonymous functions will be skipped. If called programmatically, an error is raised unless ARG is non-nil.
Move to function definition	**C-c C-f f**	(**go-goto-function** &optional ARG)	Go to the function definition (named or anonymous) surrounding point. • If we are on a docstring, follow the docstring down. • If no function is found, assume that we are at the top of a file and search forward instead. • If point is looking at the func keyword of an anonymous function, go to the surrounding function. • If ARG is non-nil, anonymous functions are ignored.

Description	Keystroke	Function	Note

Move to imports statement	`C-c C-f i`	(**go-goto-imports**)	Move point to the block of imports.
			• If using
			`import (`
			`"foo"`
			`"bar"`
			`)`
			it will move point directly behind the last import.
			• If using
			`import "foo"`
			`import "bar"`
			it will move point to the next line after the last import.
			• If no imports can be found, point will be moved after the package declaration.

Move to current method receiver	`C-c C-f m`	(**go-goto-method-receiver** &optional ARG)	Go to the receiver of the current method. • If there is none, add parenthesis to add one. • Anonymous functions cannot have method receivers, so when this is called interactively anonymous functions will be skipped. If called programmatically, an error is raised unless ARG is non-nil.
Move to current function name	`C-c C-f n`	(**go-goto-function-name** &optional ARG)	Go to the name of the current function. • If the function is a test, place point after 'Test'. • If the function is anonymous, place point on the 'func' keyword. • If ARG is non-nil, anonymous functions are skipped.
Move to current function return value declaration	`C-c C-f r`	(**go-goto-return-values** &optional ARG)	Go to the return value declaration of the current function. • If there are multiple ones contained in a parenthesis, enter the parenthesis. • If there is none, make space for one to be added. • If ARG is non-nil, anonymous functions are skipped.
Backward to beginning of function definition	• `C-M-a` • `C-M-<home>` • `<f6> <up>` • `C-[ C-a` • `Esc C-a`	(**beginning-of-defun** &optional ARG)	Move backward to the beginning of a defun. • With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun. ☛ Shift marking is available in graphics mode, not in terminal mode (for `C-M-a` and `C-M-<home>`). It's always available for `<f6> <up>` : hold Shift after typing `<f6>`.
Forward to end of function and class definition	• `C-M-e` • `C-M-<end>` • `<f6> <right>` • `C-[ C-e` • `Esc C-e`	(**end-of-defun** &optional ARG)	Move forward to next end of defun. With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun. ☛ Shift marking is available in graphics mode, not in terminal mode (for `C-M-e, C-[ C-e` and `Esc C-e` keys). However `<f6> <right>` handle Shift-marking fine in terminal mode.
Forward to start of next function definition	`<f6> <down>`	(**pel-beginning-of-next-defun** &optional SILENT DONT-PUSH_MARK)	Move forward to the beginning of the next function definition. • Beeps if does not find beginning of next function unless SILENT is non-nil. • If the beginning of next function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil. • Move back to previous position with `M-`` or `<f6><f6>`. ☛ Shift marking is available : hold Shift after typing `<f6>`.
Backward to end of previous function definition	`<f6> <left>`	(**pel-end-of-previous-defun** &optional SILENT DONT-PUSH_MARK)	Move backwards to the end of the previous function definition. • Beeps if does not find end of previous function unless SILENT is non-nil. • If the end of previous function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil. • Move back to previous position with `M-`` or `<f6><f6>`. ☛ Shift marking is available.
Indentation	See also: Σ **Indentation** generic commands that apply to go buffers. The main commands are shown here but more are available and described there.		
Indent expression at point	`C-M-q`	(**prog-indent-sexp** &optional DEFUN)	Indent the expression after point. When interactively called with prefix, indent the enclosing defun instead.
Go Syntax Checking Using either: • flycheck or • flymake See also: Σ **SyntaxCheck**	Syntax checking for the Go programming language can be done with Emacs built-in **flymake** as well as with the 📦external package **flycheck**. 🎛 • To activate either set the **pel-use-goflymake** user option is set to either 'use-flycheck or 'use-flymake. • By default, the syntax checker is not automatically launched. If you want to start your selected syntax checker as soon as a .go file is opened, add 'go-mode to the **pel-modes-activating-syntax-check** user-option. • PEL automatically installs and activates **flycheck** when **pel-use-goflymake** user option is set to 'use-flycheck. **flymake** is built-in Emacs. • 📦 Support for those is provided by the external **go-flymake.el** and **go-flycheck.el** files. PEL downloads and install them when **pel-use-goflymake** user option is set to either 'use-flycheck or 'use-flymake. • These 2 packages use the goflymake Go program, which must be installed separately. • To install the goflymake executable do the following: • Install Go on your computer if this is not already done. See instruction at the top of this page. • Set the GOPATH for your project. • Run the following command: **go get -u github.com/dougm/goflymake** • ☝ The above command will get goflymake source and install the goflymake executable file inside the bin directory of your Go project identified by the GOPATH. You will probably want to edit code in *several* Go projects, so it might be a good idea to either copy or create a symlink in one of the directories in your PATH to that file, allowing you to change GOPATH and continue to use the goflymake binary.		
Activate/deactivate selected syntax checker	`<f11> ! !`	(**pel-go-toggle-syntax-checker**)	Toggle the selected Go syntax checker mode on/off. • The syntax checker activated or deactivated is either flycheck or flymake, as selected by the user-option variable `**pel-use-goflymake**'. 📦 See the required settings above to activate this command and select the syntax checker.

Go— References

Document	Notes
Go Programming Language	• Go @ Wikipedia • Go @ home