↖ Topic Index

Emacs support for the Go Programming Language 🚧

Description	Keystroke	Function	Note
Go Support ○ Help & Customization • Info ○ Documentation/Eldoc • Navigation • Indentation ○ Syntax Check ○ Go Reference	Support for the **Go programming language** is described in this page.		
	pel-use-go Use: `<f1> o pel-use-go` to set it.		PEL supports the Go programming language when **pel-use-go** user options is set to either: • **t**: via the **go-mode** classic major mode, or • **with-tree-sitter**: via the Σ **Tree Sitter** based **go-ts-mode** when requirements are met: • Use **Emacs >= 30**, **pel-use-tree-sitter** must be set to **t**, and Go Tree Sitter grammar must be installed. See 🚦 **Tree-sitter** • 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**.
	Several aspects of Go code development is best done with Go Language Server Protocol support which can be done with **eglot** (built-in Emacs since Emacs 29.1) and the external **lsp-mode**. • Emacs **eglot** uses less resources but only supports **flymake** driven syntax check backends. Use **flycheck-eglot** to use **flycheck** syntax check backends.		
	📦 **gopls** (@GitHub)	The Go **Language Server**.	Install separately with: `go install golang.org/x/tools/gopls@latest` • Installs **gopls** in the directory identified by **$GOBIN** or **$GOPATH/bin** or in **$HOME/go/bin** • Supports: code completion, eldoc, hover, jump to def, workspace symbol, func reference, diagnostics. • Emacs LSP clients: • **Emacs Eglot** (@Github) - built-in Emacs since Emacs 29.1 • **lsp-mode** (@Github)
	PEL also provides: • Σ **Speedbar** support for .go files listing functions and types. Σ**iMenu** is supported by **go-mode** and **go-ts-mode**. • A Go section is added to Σ **Menus** by **go-mode** but nothing by **go-ts-mode**. • Generic programming language features like template text insertion handle Go comment style. See Σ **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.		
	PEL can install and activate several external packages for editing Emacs Lisp file. Activate the corresponding **pel-use-** customizable user-option:		
	📦 **flycheck-eglot**	**pel-use-flycheck-eglot**	Adds **flycheck** support for **eglot,** which by default, only supports the new **flymake**. • Use this to use =Emacs built-in **eglot** which is less resource intensive than **lsp-mode**.
	📦 **goflymake**	**pel-use-goflymake**	Supports **flycheck** syntax checking without the need for Language Server for Go. • 🚧⚠ It should support **flymake** too but I could not make it work. Anyway this would only be useful on old Emacs that cannot use a LSP client and for those **flycheck** is better than the old flymake.
	📦 **rainbow-delimiters**	**pel-use-rainbow-delimiters**	Minor-mode that highlight nested parentheses, brackets, and braces with different colours according to their depth.
See Σ **Indentation**	📦 **dtrt-indent**	**pel-use-dtrt-indent**	Detects indentation width & use of hard-tabs in file, then adjust settings to adapt.
	🎛 Format on save:	**pel-use-gofmt-on-buffer-save**	Control automatic execution of **gofmt** when saving a buffer into a file.
Last updated on:	2025-12-12		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: Σ **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.
Σ **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.
Σ **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> ? /` `<f12> ?`	(**pel-mode-setup-info** &optional APPEND) (**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.
Toggle between classic and Tree-Sitter major mode See: Σ **Tree Sitter**	`<f11> C-t C-t`	(**pel-treesit-toggle-mode**)	Toggle the major mode between the classic mode and the Tree-Sitter based mode. • If the other major mode is not available the command signals a user error.
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> <tab> w`	(**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 Σ **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.

Description	Keystroke	Function	Note
Documentation	Display code documentation based on **Emacs-lisp docstrings**. eldoc is active for all lisp-based modes. Although eldoc was primarily designed for Lisp it supports other programming languages. The function identified by **eldoc-documentation-function** is responsible for retrieving and returning the relevant documentation string for the symbol at the current point, which Eldoc then displays in the echo area. WIth Go, eldoc is best supported by a Language Server Protocol client.		
Toggle eldoc-mode Emacs Lisp Documentation Lookup	**<f12> <f4> d d**	(**eldoc-mode** &optional ARG)	Toggle echo area display of Lisp objects at point (ElDoc mode). • With a prefix argument ARG, enable ElDoc mode if ARG is positive, and disable it otherwise.

Echo area display of the Lisp object at point.	• ElDoc mode is a buffer-local minor mode. When enabled, the echo area displays information about a function or variable in the text where point is.
	  • If point is on a documented variable, it displays the first line of that variable's doc string.
	  • Otherwise it displays the argument list of the function called in the expression point is on.		

Eldoc-box	Require **eldoc-box** external package. activated by **pel-use-eldoc-box** user option. Show eldoc info in a box instead of echo area. For GUI mode only.		
Toggle eldoc-box at point	**<f12> <f4> d b**	(**eldoc-box-hover-at-point-mode** &optional ARG)	Toggle eldoc-box that displays eldoc text at point. • You can use **C-g** to hide the doc.
Toggle eldoc-box on upper corner	**<f12> <f4> d B**	(**eldoc-box-hover-mode** &optional ARG)	Displays hover documentations in a childframe. • The default position of childframe is upper corner.
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.

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: **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.

Description	Keystroke	Function	Note
Go Syntax Checking Using either: • flycheck or • flymake See also: 𝕀 **SyntaxCheck**	Syntax checking for the Go programming language can be done with Emacs built-in **flymake** as well as with the 📦external package **flycheck**. Syntax checking for Go is best supported with a Language Server Protocol client but can also be done without it on older Emacs with **flycheck**. See information at the top of this table. 👀 With PEL, as described in 𝕀 **SyntaxCheck**, you can dynamically select which syntax checking engine to use. The key bindings provided by PEL work with both engines.		
Activate/deactivate selected syntax checker	`<f11> ! !`	(**pel-fly-toggle-syntax-check** &optional *GLOBALLY*)	Toggle the current syntax check engine (**flymake** or **flycheck**) on/off. • The engine is first selected by the value of **pel-fly-engine-for-mode** user-option for the current major mode and whether **flycheck** is available (✅ available when **pel-use-flycheck** is turned on). • It changes the buffer local state of the syntax check. • You can also toggle the global state of flycheck with the optional GLOBALLY parameter. That parameter is ignored for flymake.

Go— References

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