Customizing Emacs with PEL

Operation	Keystroke	Function	Note
PEL: Control Emacs via Easy Customization • Execute `M-x pel-init` after changing configuration. • May install new packages. Last updated on:	PEL is designed to help you get going quickly with Emacs easy-to-use customization system. You do not have to write Emacs Lisp code if you don't feel comfortable doing so because PEL already has code to configure a lot of Emacs features. This table shows how to quickly gain access to the customized data using commands that open buffers that show the customized data inside buffers that operate in the Customize mode with special key bindings to speed up operation in that mode. • The first section shows navigation commands available inside a buffer that shows customized data (also called user options). • The later sections show commands that you can use to open buffers in Customization Mode to manage user options of interest. **PEL - Configuration through Customization** • PEL provides a growing set of customization groups and user option variables that control several aspects of Emacs: • The "**pel-use-**" activation user options identify what built-in or external Emacs Lisp package to use. PEL has logic to autoload the packages only when you need them. This way your Emacs will start quickly even if you have identified a large number of packages. • Once a package or feature is activated with the "pel-use-" user option, the other options control different behaviour of the activated package. 2025-10-25 ☝ After modifying the configuration, execute `M-x pel-init`. PEL will activate the new configuration.		
Generic & Specific Help & access to customization	PEL provides a set of key prefix/suffix pairs that are common to several major modes but also available globally. • The global key bindings are shown with white background in the keystroke column: from any major mode, type `<f11> <f2> <f1>` key sequences to access PEL customization specific help and `<f11> <f2> <f3>` to open PEL customization buffer. • In a customization buffer, type `<f12> <f1>` to access PEL customization specific help and `<f12> <f3>` to open PEL customization buffer.		

Open this PDF file.
See also: ⅀ **Help/Info**
	`<f11> <f2> <f1>`
	
	`<f12> <f1>`
		(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the ⅀ **Customize** 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.
			Use `<f12> <f1>` when inside a Custom-mode buffer.

Customize Emacs Customization control	`<f11> <f2> <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Customization: select how things are displayed, hooks, location of the custom file… Prefix with `C-u` to display in another window.
	`<f12> <f3>`	PEL controls the location of the custom file in the init.el. See the **pel/example/init/init.el** file. With PEL if you change it, change it in the init.el file, the value will show in the customization buffer. • Use `<f12> <f3>` when inside a Custom-mode buffer.	
Customization Data and PEL Dual Env See also: • ⅀ **Fast Startup** • **PEL user-manual**	By default Emacs stores the customization data inside the Emacs init.el file as Lisp code inside a **custom-set-variable** form. • PEL stores it inside a *separate file*, allowing dynamic selection of several files and storage into VCS independent from the init.el logic. • By default, PEL stores Emacs configuration inside **~/.emacs.d/emacs-customization.el**. • Normally Emacs uses the same configuration for Emacs running in terminal mode or graphics mode. • PEL supports the ability to use two different sets of customization files and Elpa package directories: one for Emacs running in terminal/TTY mode, another for Emacs running in graphic mode. • This feature is disabled by default. Activate it using the **pel-setup-dual-environment** command. • Type `<f11> <f2> ?` to see what is the current setup. • Type `<f11> <f2> M-d` to activate the use of the dual environment using 2 independent customization files and package directories. • When using PEL, you must place PEL-specific code inside your init.el file and inside your early-init.el file (used in Emacs ≥ 27). • PEL installation instruction describe these. • To take full advantage of PEL features, your init.el file should contain the code described in the **example/init/init.el**. • And for Emacs ≥ 27, your early-init.el should use the code described in the **example/init/early-init.el**. • Automatically create & install an early-init.el file when you activate package-quickstart with the command **pel-setup-with-quickstart**. • PEL copies the early-init.el identified by the **pel-early-init-file-template** user-option. The default is example/init/early-init.el. If you want to add logic to your early-init file, then create a file that contains the logic of example-init/early-init.el, add your own logic and identify your file inside the user-option. • ⚠️ Both init.el and early-init.el templates contain a a User Configuration section that requires manual editing. • Once created, edit the files to verify the default values of User Configuration variables and change them if required. • ☝ PEL setup commands listed in this section verify the validity of the init.el and early-init.el (if used) and will report any detected problems. • These files have identified versions. As PEL code evolves if modifications are required to these files PEL will report the required changes.		
Display state of PEL dual environment See also: ⅀ **Help/Info**	• `<f11> ? e <f2>` • `<f11> <f2> ?` • `<f11> M-S M-?`	(**pel-setup-info-dual-environment**)	Display current PEL customization setup. • Check two independent customization files for terminal/tty and graphics mode are requested and if so check if they are setup properly. • Report an error and list problems if there are any, otherwise display the current setup.
Activate PEL independent customization for Emacs in terminal/TTY mode and Emacs in graphics mode	• `<f11> <f2> M-d` • `<f11> M-S M-d`	(**pel-setup-dual-environment**)	Setup Emacs environment to support 2 independent customization. • Prompts before proceeding.
	Normally Emacs makes no distinction between those and uses the exact same set of customization files and Elpa packages for Emacs operating in those two different modes. If you want to manage the customization and packages used when Emacs operates in terminal/TTY mode one way and when Emacs operates in graphics mode another way, with PEL, then use that command. • Provide support for a customization and the Elpa directories required for the following 2 modes Emacs operation: terminal/TTY & graphics mode. • After trying to set everything for the use of dual environment it displays a message describing the state. It lists the actions performed and any remaining problems which you will have to fix manually. If all is now OK it will say so, or if all was already ok, it will also say so.		
Browse customize data tree	The following commands create a tree browser for the customize hierarchy inside a *Customize Browser* buffer. Each node can we expanded down to a single options and any can be collapsed. **Note** that PEL's customization groups and options are all always available contrary to the ones that will be available in the Emacs group because the Emacs group contains only what is currently loaded and the PEL one is always loaded.		

Browse complete customize data tree from root: Emacs	`<f11> <f2> B`
	
	`<f12> B`
		(**customize-browse** &optional GROUP)	Open the customize tree bowser for the entire Emacs customization data already loaded.
			• Unfortunately this command does not prompt it always opens the tree from the root. To specify a group use the command shown below.
			⚠️ Emacs is only able to show information it knows about. Customization data defined in files not loaded will not be accessible.

Browse customize data tree from specified group	`<f11> <f2> b`
	`<f12> b`
		(**pel-browse-group** GROUP)	Browse the customization tree from a specific group node.
			• Prompts for a group name. Supports tab completion.
			  • All PEL groups have a name that starts with "pel-".

	⚠️ Emacs can only show information it knows about. Customization data from unloaded files is not be accessible. All PEL data is always loaded. • ☝ The **pel-customize-library** commands available as the `<f3>` key of PEL key prefixes does not suffer from this problem: it will detect that the library is not loaded and will prompt you for loading it, giving you access to the customization buffer when you need it. The information is available in the various PDF pages at the top of each page.		

Browse PEL customize data tree	`<f11> <f2> P B`
	`<f12> P B`
		(**pel-browse-pel**)	Open the customize tree bowser for the entire PEL customization data (which is under Emacs/Convenience.

Customize Mode	This section describes commands available in buffer operating in Customize-mode showing the various user options you got access to using the commands described in the sections below.		
Move to Avy/Ace target (inside a customize buffer) See also:⅀ **Navigation**	`o`	(ace-link-custom)	1. Highlight each target with an Avy/Ace single or double letter target. 2. Type the letter(s) to move to that position. • This is a **very efficient and quick navigation** mechanism. 📦Requires **ace-link** 🛂 PEL activates it when **pel-use-ace-link** is set to **t**.
Apply customization changes	`C-c C-c`	(**Custom-set** &rest IGNORE)	Set the current value of all edited settings in the buffer.
Apply and Save customization changes	`C-x C-s`	(**Custom-save** &rest IGNORE)	Set all edited settings, then save all settings that have been set. • If a setting was edited and set before, this saves it. If a setting was merely edited before, this sets it then saves it.
Quit Customization and close buffer	`q`	(**Custom-buffer-done** &rest IGNORE)	Exit current Custom buffer according to 'custom-buffer-done-kill'.

Operation	Keystroke	Function	Note
Emacs Easy Customization	With the following command you can gain access to the Customize-mode to customize anything of interest. With the first command you open the customization buffer and then you can search or browse what you want to customize. The second command allow you to open the buffer at a specific customization group and the third one at a specific user option. These commands prompt for the information you are looking for. You can always use completion by typing **`<tab>`** at any point to get a list of available groups or variables. Several of the commands below open the PEL customization group and one or several other groups related to the same topic, when these groups are already loaded. • If you set the OTHER-WINDOW argument, the command open s the buffer in another window and also open any group related that exists. For example if you open the PEL group for grep with **`C-u <f11> <f2> g`**, this will also open the grep group, the rg and ripgrep groups if they are loaded. Each group will open inside its own bugger and the command will create the necessary windows. • ⚠️ **Until a package is loaded its customization group is unknown to Emacs** and no buffer will be opened for it. To see the customization group, first load the package via one of its command that is auto-loaded or load it explicitly. • ☝ Note however that the PEL commands that open customization groups attempt to identify the library where the customization group is defined and will prompt you to load the related library to enable access to the customization group. The groups accessible via the PEL commands are limited to what PEL supports.		

Customize Emacs	**`<f11> <f2> c`**
	`<f12> c`
		(**customize**)	Select a customization buffer which you can use to set user options.
			• User options are structured into "groups".
			• Initially the top-level group 'Emacs' and its immediate subgroups are shown; the contents of those subgroups are initially hidden.
			⚠️ Emacs is only able to show information it knows about. Customization data defined in files not loaded will not be accessible.

Customize a specific group	**`<f11> <f2> g`**
	`<f12> g`
		(**customize-group** &optional GROUP OTHER-WINDOW)	Customize GROUP, which must be a customization group.
			• If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window.
			• This command provides completion and you can use it to detect groups.
			⚠️ Emacs is only able to show the name names of groups that are defined in files that have already been loaded. You won't be able to open a group that is not already loaded.
			• ☝ The **pel-customize-library** commands available as the **`<f3>`** key of PEL key prefixes does not suffer from this problem: it will detect that the library is not loaded and will prompt you for loading it, giving you access to the customization buffer when you need it. The information is available in the various PDF pages at the top of each page.

Customize a user option	**`<f11> <f2> o`**
	`<f12> o`
		(**customize-option** SYMBOL)	Customize SYMBOL, which must be a user option.
			• As with groups, Emacs provides completion for user options, allowing you to detect user options.
			⚠️ Emacs is only able to show the name names of user options that are defined in files that have already been loaded. You won't be able to open a group that is not already loaded. But see the notice in the above cell.

	• **`C-h o`** • **`<f1> o`**	(**describe-symbol** SYMBOL &optional BUFFER FRAME)	Display the full documentation of SYMBOL. Can also be used to access customization buffer. Will show the info of SYMBOL as a function, variable, and/or face.

Set and store new value for user-option	**`<f11> <f2> v`**
	`<f12> v`
		(**customize-save-variable** VARIABLE VALUE &optional COMMENT)	Set the default for VARIABLE to VALUE, and save it for future sessions in the customize file.
			• Prompts for the user-option name, supporting tab completion.
			• Propose values controlled by customization selections.
			• As opposed to the commands above this does not open a customization buffer.
			☝ Use this to quickly change a PEL **pel-use-** user-option if you know its documentation and do not want to open a customization buffer.

Activate and cleanup your packages using PEL customization user-variables	PEL provides **customization-driven package management**. • PEL controls download, installation and configuration of the packages supported by its **pel-use-** user-options controlled by the PEL customization groups. The packages missing are installed when you start Emacs or when you explicitly run the **pel-init** command. • PEL also removes the packages that are not required by the PEL user-options when you issue the **pel-cleanup** command. • Use a key prefix for this command to perform a dry-run of the command and produce a report of what would be removed. • PEL does not delete packages. Instead it places them into separate directories, called "attic" directories. You can then retrieve the package from the directories later. • The elpa packages are stored in the directory identified by **package-user-dir** or in the "elpa" directory inside the **user-emacs-directory**. • The elpa attic is identified by a name that appends "-attic" to the above directory name. • On a Unix-like system that would normally be "~/.emacs.d/elpa" and "~/.emacs.d/elpa-attic". • The non-elpa files are stored in the directory identified by the **pel-utils-dirname** user-options (which defaults to "utils") inside the directory identified by the **user-emacs-directory**. Its attic directory name is the same name with a "-attic" suffix. • By default, on Unix-like systems the directories are "~/.emacs.d/utils" and "~/.emacs.d/utils-attic". • On Windows system the directories are located in your User directory, as controlled by Emacs. Also on Emacs 27.1 and later these directories can be located somewhere else. • The customization file (described in a section above) should be located in the same **user-emacs-directory**.		
Re-initialize PEL, activate the new PEL user-option, install packages newly requested	`M-x pel-init`	(**pel-init** &optional CACHED-ABBREV-FILE-NAME)	Re-initialize PEL. Download, install and configure any package requested by the various **pel-use-** user-options that have not yet been installed. • Does not remove anything. Use pel-cleanup for that. • The argument is not accessible interactively and exists for the initial Emacs startup only.
Show PEL user option and package info See also: ∑ **Help/Info**	**`<f11> ? e ?`**	(**pel-package-info** &optional FULL-REPORT ON-STDOUT	Display the following information inside a *pel-user-options* buffer: • name of custom file, package-user-dir, the number of PEL user-options, and the number of them that are active, number of loaded files, and features. • The number of Elpa packages active: the count of the ones directly installed because of active PEL user-options and the count of them installed as dependencies of the first group. • The number of Emacs Lisp files stored in the ~/.emacs.d/utils (or equivalent directory) as a result of PEL user options. • The number of elpa-compliant packages that have a newer version and could be updated. • With optional argument, like **`C-u`**, generates a full report with more details.
Disable all packages not requested by PEL user-options and not identified as dependency or packages that must be kept. **Update the load path and the customization file content.**	`M-x pel-cleanup`	(**pel-cleanup** &optional DRY-RUN)	After prompting for a confirmation, de-activates all Elpa and non-Elpa packages that are not requested by a PEL user-option. The command keeps packages that are dependencies of packages required by PEL user-options and packages that PEL always requires. It also keeps packages that you have identified as manually installed in the following user options: • **pel-elpa-packages-to-keep** • **pel-utils-packages-to-keep** ⚠️ For the current version of PEL when you install an Emacs package with the Emacs package system, PEL does not automatically add the package name in the **pel-elpa-packages-to-keep** user-option. If you want to keep that package and configure it yourself with your own Emacs Lisp code invoked by your init.el file, add the package symbol name to the list of **pel-elpa-packages-to-keep** otherwise **pel-cleanup** will move the package to the elpa-attic.
Perform a dry-run of pel-cleanup. **Generate a detailed report.**	`M-- M-x pel-cleanup`		Runs **pel-cleanup** in dry-mode and produce a detailed report of what **pel-pel-cleanup** would remove in a *pel-cleanup* buffer.

Operation	Keystroke	Function	Note
Input Completion Mode Selection See also: • ∑ Completion/Input • ∑ Menus • ∑ Navigation	PEL supports several input completion modes that kick in with the `M-x`, `C-x b`, `C-x C-f`, `<f1> o` and many other commands. PEL supports the following input completion modes: 1. Emacs' default tab completion 2. **Helm mode** completion : set **pel-use-helm** to **t**. 3. **Ido mode** completion : set **pel-use-ido** to **t** 4. **Ivy mode** completion : set **pel-use-ivy** to **t** 5. **Ivy mode completion with Counsel mode** : set **pel-use-counsel** to **t** 6. Ido/Helm mode where Ido is used for dealing with Files and buffers and Helm is used everywhere else (including all Helm specific commands). • PEL also has commands that uses the iMenu system to list symbol defined in the current or all buffers. The behaviour and user interface or these commands can be modified and extended by several external packages and customization user-options: • **pel-imenu-follows-order-p** user-option controls whether entries are sorted or follows the order of declaration in the file. • **flimenu** external package activated by **pel-use-flimenu** user-option, controls whether iMenu lists are flatten or hierarchical. • **imenu-anywhere** external package activated by **pel-use-imenu-anywhere** user-option is used by **pel-goto-symbol-any-buffer** to jump to symbol definition of any buffer using one of the following input completion method. The user-option must be set to one of the following values: • Use emacs-default: basic Emacs completion. Use tab to see possible matches. • Use Ido. **pel-use-ido** must be turned on. • Use Ivy. Requires **Ivy mode** **pel-use-ivy** must be on. • Use helm. Requires **Helm mode** **pel-use-helm** must be turned on. • **popup-imenu** external package activated by **pel-use-popup-imenu** user-option, provides one pop-up menu for the iMenu content. • **popup-switcher** external package activated by **pel-use-popup-switcher** user-option, provides the same as popup-imenu and more. To customize the above, use: • `<f11> M-c <f2>` to customize the PEL completion group user options. It is also available via `M-g <f4> <f2>`. • `<f11> <f10> <f2>` to customize the PEL iMenu user-options. As soon as one of the extra completion mode is activated via the corresponding pel-use- user option, PEL makes the following commands available to change the completion mode and to see which one is currently active.		
Select the completion mode	`<f11> M-c <f4>`	(pel-select-completion-mode)	Prompt user for completion mode to activate. The available modes depend on what is currently activated by customization. See the list above.

Show what completion mode is currently used.	`<f11> M-c ?`
	`<f11> ? M-c`
		(pel-show-active-completion-mode)	Display the completion mode currently used, and the Ido prompt geometry when appropriate. Show key bindings for changing other aspects of input completion.

Search Tools Selection See also: ∑ Search/Replace	PEL supports several search tools that impact the way the `C-s` command operates. PEL supports the following search tools: • Emacs' default ISearch • Anzu, ISearch with match count : set **pel-use-anzu** to **t**. • Swiper search with overview match list : set **pel-use-swiper** to **t** Use `<f11> s <f3>` to customize the PEL completion group user options above. • Set the **pel-initial-search-tool** user option to select which search tool is used when Emacs starts. As soon as one of the extra search tool is activated via the corresponding pel-use- user option, PEL makes the following commands available to change the currently used search tool and to see which one is currently active.		
Show which search tool is currently used	`<f1> ? s`	(pel-show-active-search-tool)	Display the currently used search tool.
Select search tool to use	`<f11> s s`	(pel-select-search-tool)	Prompt user for search tool to use with `C-s`. Show new active one.
	• Emacs normally maps the search-forward command to `C-s`. • PEL provides the ability to activate the following tools that can be activated for searching: • The Anzu external package activated by **pel-use-anzu** user option. Anzu provides a match count in the mode line when searching. • The Swiper external package activated by **pel-use-swiper** user option. Swiper is not using isearch-forward; it shows a list of matching lines in the mini-buffer. • Use the `<f11> s <f2>` command to open the PEL search customize group and set the **pel-initial-search-tool** user option to identify which tool is used when Emacs starts. ☝ Being able to search using either Emacs default ISearch (see below) and Swiper helps as they are both very useful in different scenarios.		
Customize PEL support ☝ ⚠	The following commands opens the Emacs customization group related to a PEL topic. Most of these commands do not prompt; they open the customization buffer at the requested group. • If you prefix the following commands with `C-u` PEL also opens the customization groups related to the specific feature. • To activate any PEL customization change in the current session, execute `M-x pel-init` after you saving and applying the customized variable. For motion variables that control mode hooks (eg. the flyspell automatic activation for specific major modes), also restart Emacs.		

All PEL	`<f11> <f2> P !`
	`<f12> P !`
		(pel-cfg &optional OTHER-WINDOW)	Customize PEL support.
			• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.

PEL base	`<f11> <f2> p`
	`<f12> p`
		(pel-customize-pel-base-emacs-group &optional OTHER-WINDOW)	Customize basic PEL configuration: open the **pel-base-emacs** group.
			• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.

Customize specific PEL group	The following key bindings almost all use the same PEL command: (**pel-customize-pel** &optional OTHER-WINDOW). The command detects the key sequence that invoked it to select the customization group to open. If there are more than one it prompts for the one to open. If a group is not loaded, PEL prompts for loading it. • All of these commands open the buffer inside another window if a prefix argument (like `C-u)` is typed first.		
∑ **Align**	`<f11> t a <f2>`	Customize PEL support for text alignment.	
∑ **Auto-Completion**	`<f11> , <f2>`	Customize PEL auto-completion support: auto-complete, company and hippie-expand.	
∑ **Bookmarks**	`<f11> ' <f2>`	Customize PEL support for bookmark groups: bookmark, bm.	
∑ **Buffers**	`<f11> b <f2>`	Customize PEL support for buffer management: hexl.	
∑ **Comments**	`<f11> ; <f2>`	Customize Emacs support for comment hide control: hide-cmnt.	
∑ **Cursor**	`<f11> m <f2>`	Customize PEL support for cursor and multiple-cursors.	
∑ **Filling/Justification**	• `<f11> t f <f2>` • `<f11> t j <f2>`	Customize PEL support for:	
∑ **Diff & Merge**	`<f11> d <f2>`	Customize PEL support for diff: ztree.	
∑ **Dired**	`<f11> f <f2> 2`	Customize PEL support for dired, directory editor.	
∑ **Drawing**	`<f11> D <f2>`	Customize PEL drawing mode support.	
∑ **Fast Startup**	`<f11> M-S <f2>`	Customize PEL support for fast startup mode.	
∑ **File-mngt**	`<f11> f <f2> 1`	Customize PEL support for file management.	
∑ **File-mngt** - dir. tree browser	`<f11> B <f2>`	Customize PEL support for directory tree browsers, web browser and ztree	
∑ **File-mngt** - NeoTree	`<f11> B N <f2>`	Customize PEL support for NeoTree directory browser	
∑ **Frames**	`<f11> F <f2>`	Customize PEL frame management support.	

Operation	Keystroke	Function	Note
∑ Grep	`<f11> g <f2>`	Customize PEL grep support. Groups: grep, ag, rg, ripgrep, wgrep.	
∑ Help/Info	`<f11> ? <f2>`	Customize PEL help support.	
∑ Hide/Show	`<f11> M-/ <f2>`	Customize PEL support for comments: hide-cmnt, hide-lines.	
∑ Highlight	`<f11> h <f2>`	Customize PEL support for buffer highlight management: fill-column-indicator, vline, rainbow-delimiters.	
∑ Indentation	`<f11> <tab> <f2>`	Customize PEL support for:	
∑ Inserting Text	`<f11> i <f2>`	Customize PEL text insertion support: lice, smart-dash, tempo, time-stamp, yasnippet	
∑ Keyboard Macros	• `<f11> k <f2>` • `<f11> k e <f2>` • `<f11> k l <f2>`	Customize the PEL keyboard macro external package support: centimacro, emacros, elmacro.	
∑ Key-Chords	`<f11> <f5> k <f2>`	Customize PEL Key Chord support.	
Input Completion: ∑ Completion/Input	• `<f11> M-c <f2>` • `M-g <f4> <f2>`	Customize PEL Input Completion support.	
∑ Marking	`<f11> . <f2>`	Customize PEL Marking support.	
∑ Menus - **iMenu**	`<f11> <f10> <f2>`	Customize PEL imenu support.	
∑ Mode Line	`<f11> M-d <f2>`	Customize PEL mode line support	
∑ Navigation	`<f11> <f2> P n`	(**pel-cfg-pkg-navigation** &optional OTHER-WINDOW)	Customize PEL and Emacs navigation tools support. Provides access to the following customization groups: 1. **PEL project management** 2. **avy** • If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window.
∑ Outline	`<f11> SPC M-l <f2>`	Customize PEL outline support	
∑ Projectile	• `<f11> <f2> P <f8>`	(**pel-cfg-pkg-project-mng** &optional OTHER-WINDOW)	Open the projectile customization group where you can modify projectiles configuration. • The key sequence `<f11> <f2> P <f8>` is always available, the others are only available when the projectile mode is activated. 📦 Available when the **projectile** external package is ☑ activated by PEL with the **pel-use-projectile** user option is non-nil.
	• `<f11> <f8> <f2>` • `<f8> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	
∑ Scrolling	`<f11> | <f2>`	Customize PEL Scrolling support.	
∑ Search/Replace	`<f11> s <f2>`	Customize PEL basic search support.	
Regular Expression ∑ Search/Replace	`<f11> s x <f2>`	Customize PEL regular expression tool support.	
∑ Sessions	`<f11> S <f2>`	Customize PEL Session support.	
∑ Shells	`<f11> z <f2>`	Customize PEL Shell support.	
∑ Speedbar	`<f11> M-s <f2>`	Customize PEL Speedbar support.	
∑ Spell Checking	`<f11> $ <f2>`	Customize PEL support for: spell checking. Identify which major modes will automatically activate either flyspell-mode or flyspell-prog-mode.	
∑ Text Modes	• `<f11> t <f2>` • `<f11> t m <f2`	Customize PEL text management support.	
∑ Time Tracking	`<f11> T <f2>`	Open the PEL customize group(s) for the current context.	
∑ Undo/Redo/Repeat/Arg	`<f11> u <f2>`	Customize PEL undo support.	
∑ VCS	`<f11> v <f2>`	Customize PEL Version Control System support.	
∑ Windows	`<f11> w <f2>`	Customize PEL Window support.	
Yasnippet - ∑ Inserting Text	`<f11> y <f2>`	Customize PEL Yasnippet text insertion support.	
∑ Xref **- cross reference**	`<f11> X <f2>`	Customize PEL cross-reference support: ctags/etags/gtags	

Operation	Keystroke	Function	Note
Customize PEL Programming Language support	The following commands opens the Emacs configuration group to configure PEL support for the specified programming language. • You should be able to control most of the important features of the programming languages through these customizations including the activation of important packages as well as aspects of programming language styles like indentation style and width. • The `<f11> SPC` key prefixes are available globally (for all buffers). • The `<f12> <f2>` key is only available when point is in a buffer for one of the languages supported by PEL and open the PEL customization group for the programming language for the current buffer. ☝ When you use the `<f11> SPC` prefix, you can customize the Emacs language library support that might not even be loaded: PEL will detect if the corresponding library is loaded and will prompt you asking if you want to load it first, allowing Emacs to open the customization buffer. ⚠️ ☝To activate any PEL customization change in the current session, execute `M-x pel-init` after you saving and applying the customized variable. Alternatively close and re-start Emacs.		

AppleScript & text audio narration	`<f11> SPC a <f2>`
	`<f12> <f2>`
		Customize PEL Applescript support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Arc	`<f11> SPC C-a <f2>`
	`<f12> <f2>`
		Customize PEL Arc support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - C	`<f11> SPC c <f2>`
	`<f12> <f2>`
		Customize PEL C support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - C++	`<f11> SPC C <f2>`
	`<f12> <f2>`
		Customize PEL C++ support: cpp.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Clojure	`<f11> SPC C-j <f2>`
	`<f12> <f2>`
		Customize PEL Clojure support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Common Lisp	`<f11> SPC L <f2>`
	`<f12> <f2>`
		Customize PEL Lisp support: lisp, lispy.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Chez Scheme	`<f11> SPC C-s C-z <f2>`
	`<f12> <f2>`
		Customize PEL Chez support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Chibi Scheme	`<f11> SPC C-s C-i <f2>`
	`<f12> <f2>`
		Customize PEL Chibi support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Chicken Scheme	`<f11> SPC C-s C-k <f2>`
	`<f12> <f2>`
		Customize PEL Chicken support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - D	`<f11> SPC D <f2>`
	`<f12> <f2>`
		Customize PEL D support: d-mode.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Elixir	`<f11> SPC x <f2>`
	`<f12> <f2>`
		Customize PEL Elixir support: alchemist, alchemist-iex.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔏𝔓𝔩 - Emacs Lisp	`<f11> SPC l <f2>`
	`<f12> <f2>`
		Customize PEL Elisp support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔏𝔓𝔩 - Emacs Lisp eldoc	`<f11> SPC l ? <f2>`
	`<f12> <f2>`
		Customize PEL Elisp support: eldoc-box.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Erlang	`<f11> SPC e <f2>`
	`<f12> <f2>`
		Customize PEL Erlang support: erlang, erldoc, edts, auto-highlight-symbol.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Forth	`<f11> SPC f <f2>`
	`<f12> <f2>`
		Customize PEL Forth support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Go	`<f11> SPC g <f2>`
	`<f12> <f2>`
		Customize PEL Go support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Gambit Scheme	`<f11> SPC C-s C-b <f2>`
	`<f12> <f2>`
		Customize PEL Gambit Scheme support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - GNU Guile Scheme	`<f11> SPC C-s C-g <f2>`
	`<f12> <f2>`
		Customize PEL Guile support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Gerbil Scheme	`<f11> SPC C-s C-e <f2>`
	`<f12> <f2>`
		Customize PEL Gerbil Scheme support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Gleam	`<f11> SPC M-G <f2>`
	`<f12> <f2>`
		Customize PEL Gleam support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Haskell	`<f11> SPC h <f2>`
	`<f12> <f2>`
		Customize PEL Haskell support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Hy	`<f11> SPC C-h <f2>`
	`<f12> <f2>`
		Customize PEL Hy support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Julia	`<f11> SPC j <f2>`
	`<f12> <f2>`
		Customize PEL Julia support: julia, julia-mode, julia-snail.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - Janet	`<f11> SPC T <f2>`
	`<f12> <f2>`
		Customize PEL Janet support: pel-use-janet, pel-use-janet-mode, pel-use-ijanet, pel-use-inf-janet
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - LFE	`<f11> SPC C-l <f2>`
	`<f12> <f2>`
		Customize PEL LFE support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩- Lispy	`<f11> <f2> SPC M-L`	Customize support for Lisp programming languages - A group that also contains the groups for Emacs Lisp and Common Lisp: lispy. • If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

𝔓𝔩 - NetRexx	`<f11> SPC N <f2>`
	`<f12> <f2>`
		Customize PEL NetRexx support. Use this to activate NetRexx support.
		• If OTHER-WINDOW is non-nil (use `C-u`), display in another window.	

Operation	Keystroke	Function	Note
𝔓𝔩 - Nim	`<f11> SPC n <f2>`	Customize PEL nim support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - OCaml	`<f11> SPC o <f2>`	Customize PEL OCaml support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - Perl	`<f11> SPC P <f2>`	Customize PEL Perl support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - Python	`<f11> SPC p <f2>`	Customize PEL Python support: python, python-flymake. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - Racket	`<f11> SPC C-s C-r <f2>`	Customize PEL Racket support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - REXX	`<f11> SPC R <f2>`	Customize PEL REXX support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - Ruby	`<f11> SPC U <f2>`	Customize PEL Ruby support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - Rust	`<f11> SPC r <f2>`	Customize PEL Rust support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - UNIX Shell	`<f11> SPC H <f2>`	Customize PEL UNIX Shell support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - Scheme	`<f11> SPC C-s C-s <f2>`	`<f11> SPC C-s C-s <f2>` `<f12> <f2>`	
	`<f12> <f2>`		
𝔓𝔩 - V	`<f11> SPC v <f2>`	Customize PEL V support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
Customize PEL Markup support	The following commands opens the Emacs customization group related to configure PEL support for the specific markup language. • The **`<f11> SPC`** key prefixes are available globally (for all buffers). • The **`<f12> <f2>`** key is only available when point is in a buffer for one of the languages supported by PEL and open the PEL customization group for the markup language for the current buffer. ⚠️ ☝️To activate any PEL customization change in the current session, execute **`M-x pel-init`** after you saving and applying the customized variable. ⚠️ ☝️To activate any PEL customization change in the current session, execute **`M-x pel-init`** after you saving and applying the customized variable. Alternatively close and re-start Emacs.		
Ɱ Graphviz Dot	`<f11> SPC M-g <f2>`	Customize PEL Graphviz-Dot support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
Ɱ PlantUML	• `<f11> D u <f2>` • `<f11> SPC M-u <f2>`	Customize PEL PlantUML support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
Ɱ Markdown	`<f11> SPC M-m <f2>`	Customize PEL Markdown support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
Ɱ Outline/Org-Mode	`<f11> SPC M-o <f2>`	Customize PEL Org Mode support: open pel-pkg-for-org-mode group. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
Ɱ reStructuredText	`<f11> SPC M-r <f2>`	Customize PEL reStructuredText support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
Customize Specific Emacs Groups.	PEL provides several key bindings to open customization groups of Emacs built-in or external package. • PEL will prompt you to load their specific file if they are not loaded. • Most of the key bindings are mapped into the PEL key prefixes as the **`<f3>`** key member. For example to open auto-completion related groups you can use the **`<f11> , <f3>`** key sequence. These are not listed here. • PEL does not provide key prefixes for all Emacs concepts. It provides, however some key bindings to access the customization buffer for some of those. They are listed just below, here:		
Permanently change the cursor's color See also: ⅀ **Cursor**	`<f11> <f2> E C-c`	(**pel-customize-cursor** &optional OTHER-WINDOW)	Quicks access to the customize buffer to set the cursor default color. • It sets the color permanently if the customization is saved. ⚠️ Only available in graphics mode.
locate	`<f11> <f2> E l`	(**pel-cfge-locate** &optional OTHER-WINDOW)	Customize locate. With **C-u**, display in another window.
man	`<f11> <f2> E m`	(**pel-cfge-man** &optional OTHER-WINDOW)	Customize man. With **C-u**, display in another window.
browse-url	`<f11> <f2> E u`	(**pel-cfge-browse-url** &optional OTHER-WINDOW)	Customize browse-url. With **C-u**, display in another window.
webjump	`<f11> <f2> E j`	(**pel-cfge-webjump** &optional OTHER-WINDOW)	Customize webjump. With **C-u**, display in another window.
woman	`<f11> <f2> E w`	(**pel-cfge-woman** &optional OTHER-WINDOW)	Customize woman. With **C-u**, display in another window.
Customize Emacs Libraries	The following key bindings almost all use the same PEL command: (**pel-customize-library** &optional OTHER-WINDOW). The command detects the key sequence that invoked it to select the customization group to open. If there are more than one it prompts for the one to open. If a group is not loaded, PEL prompts for loading it. If the related package is not installed PEL print a warning message. • For external packages you can use the same key sequence except for the last key: replace **`<f3>`** by **`<f2>`** : that sequence will open the PEL configuration buffer for the same topic. From that you will find the PEL option variable to activate the external package. • All of these commands open the buffer inside another window if a prefix argument (like **`C-u)`** is typed first.		
⅀ Align	`<f11> t a <f3>`	Customize Emacs text alignment support: open the align group.	
⅀ Auto-Completion	`<f11> , <f3>`	Customize Emacs auto-completion support: auto-complete, company and hippie-expand.	
⅀ Bookmarks	`<f11> ‘ <f3>`	Customize Emacs bookmark group which includes: bookmark and bm.	
⅀ Buffers	`<f11> b <f3>`	Customize Emacs support for buffer management: Buffer-menu, buffer, minibuffer, hexl, nhexl.	
⅀ Comments	`<f11> ; <f3>`	Customize Emacs support for comments: comment, hideshow.	
Customization Control	`<f11> <f2> <f3>`	Customize Emacs customization control.	
⅀ Hide/Show	`<f11> M-/ <f3>`	Customize Emacs support for comments: comment, hideshow.	

Operation	**Keystroke**	**Function**	**Note**
Input Completion: **∑ Completion/Input**	`<f11> <f2> P M-c`	**(pel-cfg-pkg-completion** &optional OTHER-WINDOW)	Customize Emacs Input Completion support: helm, ido, ivy, counsel.. • If OTHER-WINDOW is non-nil (use **`C-u`**), display in other window.
∑ Cursor	`<f11> m <f3>`	Customize Emacs support for cursor and multiple-cursors.	
∑ Diff & Merge - ediff	`<f11> d e <f3>`	Customize Emacs ediff.	
∑ Dired	`<f11> SPC M-D <f3>` `<f12> <f3>`	Customize Emacs support for: dired, dired-git-info, dired-hide-dotfiles, ls-lisp, wdired. • The **`<f12> <f3>`** key sequence is available in the dired buffer.	
∑ Enriched Text	`<f11> t e <f3>`	Customize Emacs Enriched Text support.	
∑ File-mngt	`<f11> f <f3> l`	Customize Emacs support for file management.	
∑ File-mngt - auto-revert	`<f11> f r <f3>`	Customize Emacs support for file automatic revert management.	
∑ File-mngt - ffap	`<f11> f a <f3>`	Customize Emacs support for management of ffap (find file at point).	
∑ File-mngt - dir. tree browser	`<f11> B <f3>`	Customize directory tree browsers: dir-treeview, lsp-treemacs, rfc-mode-group, treemacs, ztree	
∑ File-mngt - NeoTree	`<f11> B N <f3>`	Customize NeoTree directory browser	
∑ Filling/Justification	• `<f11> t f <f3>` • `<f11> t j <f3>`	Customize Emacs fill and justification control.	
∑ Frames	`<f11> F <f3>`	Customize Emacs frame management support.	
∑ Grep	`<f11> g <f3>`	Customize Emacs grep support. Groups: grep, ag, deadgrep, fzf, rg, ripgrep, wgrep.	
∑ Help/Info	`<f11> ? <f3>`	Customize Emacs help support. Groups: command-log, helpful.	
∑ Highlight	`<f11> h <f3>`	Customize Emacs support for buffer highlight management: auto-highlight, edit, rainbow-delimited, line, fill-column-indicator (for Emacs version earlier than 27.1)	
∑ Indentation	`<f11> <tab> <f3>`	Customize Emacs indentation. Opens the indent customization group.	
∑ Inserting Text	`<f11> i <f3>`	Customize Emacs text insertion support: lice, smart-dash, tempo, time-stamp, yasnippet	
∑ Keyboard Macros	`<f11> k <f3>`	Customize the Emacs keyboard macro external package support: kmacro, centimacro.	
∑ Keyboard Macros	`<f11> k e <f3>`	Customize the Emacs keyboard macro external package support: emacros.	
∑ Keyboard Macros	`<f11> k l <f3>`	Customize the Emacs keyboard macro external package support: elmacro.	
∑ Key-Chords	`<f11> <f5> k <f3>`	Customize Emacs support for: key-chord	
Line Mngt: **∑ Display - Lines**	`<f11> l <f3>`	Customize Emacs support for visual-line.	
∑ Marking	`<f11> . <f3>`	Customize Emacs Marking support.	
∑ Menus - iMenu	`<f11> <f10> <f3>`	Customize Emacs menu mechanisms.	
∑ Mode Line	`<f11> M-d <f3>`	Customize Emacs mode line support: mode-line	
∑ Navigation	`<f11> <f2> P n 2`	**(pel-cfg-pkg-navigation** &optional OTHER-WINDOW)	Customize Emacs navigation tools support: avy. • If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window.
∑ Outline	`<f11> SPC M-l <f3>`	Customize Emacs outline support	
∑ Projectile	• `<f11> <f8> <f3>` • `<f8> <f3>`	**(pel-customize-projectile)**	Open the projectile customization group where you can modify projectiles configuration.
		• Key sequence **`<f11> <f8> <f3>`** is available if **pel-use-projectile** is **t**. • Key sequence **`<f8> <f2>`** is available when the projectile mode is on. 📦 Available when **projectile** external package is ☑ activated the **pel-use-projectile** user option.	
Regular Expression **∑ Search/Replace**	`<f11> s x <f3>`	Customize Emacs regular expression support: rxt, re-builder, visual-regex.	
∑ Scrolling	`<f11> | <f3>`	Customize Emacs Scrolling support groups: follow, smooth-scrolling.	
∑ Search/Replace	`<f11> s <f3>`	Customize Emacs Search support: isearch, anzu, iedit, easy-escape, fzf, swiper.	
∑ Sessions	`<f11> S <f3>`	Customize Emacs Session support: desktop.	
∑ Shells	`<f11> z <f3>`	Customize Emacs Shells support groups: term, terminals, vterm.	
∑ Speedbar	`<f11> M-s <f3>`	Customize Emacs Speedbar support.	
∑ Spell Checking	`<f11> $ <f3>`	Customize Emacs spelling support. Opens the following customization groups: ispell, flyspell.	
∑ Text Modes	`<f11> t m <f3>`	Customize Emacs text mode group: **glasses**	
Text ∑ Whitespace	`<f11> t w <f3>`	Customize Emacs handling of whitespaces.	
∑ Time Tracking	`<f11> T <f3>`	Customize Emacs *time related* groups which includes: display-time, timeclock, timelog	
∑ VCS	`<f11> v <f3>`	Customize Emacs Version Control System support: vc, vc-hg, vc-git, magit, monky.	
∑ Undo/Redo/Repeat/Arg	`<f11> u <f3>`	Customize Emacs undo support: undo, undo-tree.	
∑ Windows	`<f11> w <f3>`	Customize Emacs Window support groups: windows, ace-window, ace-window-display, winner, windmove.	
∑ Xref - cross reference	`<f11> X <f3>`	Customize Emacs cross-reference support: ctags/etags/gtags	
Yasnippet **∑ Inserting Text**	`<f11> y <f3>`	Customize Yasnippet groups: yasnippet, yasnippet-snippets, yas-minor	

Operation	Keystroke	Function	Note
Customize Emacs Programming Language support	The following commands opens the Emacs configuration group to configure **Emacs** support for the specified programming language. • The `<f11> SPC` key prefixes are available globally (for all buffers). • The `<f12> <f3>` key is only available when point is in a buffer for one of the languages supported by PEL and open the Emacs customization group for the programming language for the current buffer. ☝ When you use the `<f11> SPC` prefix, you can customize the Emacs language library support that might not even be loaded: PEL will detect if the corresponding library is loaded and will prompt you asking if you want to load it first, allowing Emacs to open the customization buffer.		
AppleScript & text audio narration	`<f11> SPC a <f3>`	Customize Emacs Applescript support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Arc	`<f11> SPC C-a <f3>`	Customize Emacs Arc support: arc, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - C	`<f11> SPC c <f3>`	Customize Emacs C support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - C++	`<f11> SPC C <f3>`	Customize Emacs C++ support: cpp. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Clojure	`<f11> SPC C-j <f3>`	Customize Emacs Clojure support: clojure, cider, cljr. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Common Lisp	`<f11> SPC L <f3>`	Customize Emacs Lisp support: lisp, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Chez Scheme	`<f11> SPC C-s C-z <f3>`	Customize Emacs Scheme support: scheme, geiser, quack, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Chibi Scheme	`<f11> SPC C-s C-i <f3>`	Customize Emacs Scheme support: scheme, geiser, quack, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Chicken Scheme	`<f11> SPC C-s C-k <f3>`	Customize Emacs Scheme support: scheme, geiser, quack, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - D	`<f11> SPC D <f3>`	Customize Emacs D support: d-mode. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Elixir	`<f11> SPC x <f3>`	Customize Emacs Elixir support: alchemist, alchemist-iex. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔏𝔓𝔩 - Emacs Lisp	`<f11> SPC l <f3>`	Customize Emacs Elisp support: checkdoc, editing-basics, elint, eldoc, eros, lisp, lispy, suggest. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔏𝔓𝔩 - Emacs Lisp eldoc	`<f11> SPC l ? <f3>`	Customize PEL Elisp support: eldoc, eldoc-box. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Erlang	`<f11> SPC e <f3>`	Customize Emacs Erlang support: erlang, erldoc, edts, auto-highlight-symbol. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Forth	`<f11> SPC f <f3>`	Customize Emacs Forth support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Go	`<f11> SPC g <f3>`	Customize Emacs Go support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f2>`		
𝔓𝔩 - Gambit Scheme	`<f11> SPC C-s C-b <f3>`	Customize Emacs Scheme support: gerbil-mode, scheme, geiser, quack, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - GNU Guile Scheme	`<f11> SPC C-s C-g <f3>`	Customize Emacs Scheme support: scheme, geiser, quack, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Gerbil Scheme	`<f11> SPC C-s C-e <f3>`	Customize Emacs Scheme support: gerbil-mode, scheme, geiser, quack, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Haskell	`<f11> SPC h <f3>`	Customize Emacs Haskell support: haskell • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Julia	`<f11> SPC j <f3>`	Customize Emacs Julia support: julia, julia-mode, julia-snail. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Janet	`<f11> SPC T <f3>`	Customize Emacs Janet support: janet, ijanet, inf-janet • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - LFE	`<f11> SPC C-l <f3>`	Customize Emacs LFE support: the **lfe** customization group, which controls the settings of the lfe-mode. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Make	`<f11> SPC M <f3>`	Customize Emacs makefile support: makefile. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - NetRexx	`<f11> SPC N <f3>`	Customize Emacs NetRexx support: netrexx-mode • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Nim	`<f11> SPC n <f3>`	Customize Emacs nim support: nim • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - OCaml	`<f11> SPC o <f3>`	Customize Emacs OCaml support: merlin, tuareg, tuareg-opam. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Perl	`<f11> SPC P <f3>`	Customize Emacs Perl support: perl. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔩 - Python	`<f11> SPC p <f3>`	Customize Emacs Python support: python, python-flymake. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		

Operation	Keystroke	Function	Note
𝔓𝔏 - Racket	`<f11> SPC C-s C-r <f3>`	Customize Emacs Racket support: racket, scheme, geiser, quack, lispy. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔏 - REXX	`<f11> SPC R <f3>`	Customize Emacs REXX support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔏 - Ruby	`<f11> SPC U <f3>`	Customize Emacs Ruby support: ruby. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔏 - Rust	`<f11> SPC r <f3>`	Customize Emacs Rust support: rust-mode, rustic, racer, cargo. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔏 - Scheme	`<f11> SPC C-s C-s <f3>`	Customize PEL Scheme support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔏 - UNIX Shell	`<f11> SPC H <f3>`	Customize Emacs UNIX Shell support: sh, sh-script, sh-indentation. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
𝔓𝔏 - V	`<f11> SPC v <f3>`	Customize Emacs V support: v • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
Customize Emacs Markup support	The following commands opens the Emacs customization group related to configure **Emacs** support for the specific markup language. • The **`<f11> SPC`** key prefixes are available globally (for all buffers). • The **`<f12> <f3>`** key is only available when point is in a buffer for one of the languages supported by PEL and open the Emacs customization group for the markup language for the current buffer. ☝️ When you use the **`<f11> SPC`** prefix, you can customize the Emacs language library support that might not even be loaded: PEL will detect if the corresponding library is loaded and will prompt you asking if you want to load it first, allowing Emacs to open the customization buffer.		
Ɱ Graphviz Dot	`<f11> SPC M-g <f3>`	Customize Emacs Graphviz-Dot support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
Ɱ PlantUML	• `<f11> D u <f3>` • `<f11> SPC M-u <f3>`	Customize Emacs PlantUML support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
Ɱ Markdown	`<f11> SPC M-m <f3>`	Customize Markdown and markdown extension package support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		
Ɱ Outline/Org-Mode	`<f11> SPC M-o <f3>`	Customize Org Mode external packages support: • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
Ɱ reStructuredText	`<f11> SPC M-r <f3>`	Customize Emacs reStructuredText support. • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.	
	`<f12> <f3>`		