↖ Topic Index

rst-mode: reStructuredText Mode

Description	Keystroke	Function	Note
reStructuredText ○ Help & customize • rst-mode, syntax-control • Fill text, emphasis (bold,...) • itemize list of lines ○ Format reStucturedText table • View/Navigate Table of Contents • Insert file header ○ Adorn Section Level • Creating/Using hyperlinks • Using goto-url-mode ○ Copy URL target in file & visit it • Open File at point • Supports reStructured text hyperlinks • compile/render file **More Info on reSTructuredText:** Last updated on:	This page describes Emacs support for reStructuredText (abbreviated sometimes as '*rst*' and sometimes as '*reST*') . • The reSructuredText files are supported by Emacs rst-mode from rst.el which is available in standard Emacs distribution. • **Supported file extensions**: .rst, .rest, .stxt and .rst.txt. The .rst.txt extension allows rendering by tools supporting .txt files. To activate it under PEL, you must set the PEL **pel-use-rst** customization variable to **t**. **pel-rst-tab-width**: The width of a tab used for reStructuredText files. Defaults to 2. • This concept differs from indentation: you can have an indentation of 3 and tab width of 8: **M-i** will move point to columns that are multiple of 8 **<tab>** will indent to a column that is a multiple of 3. PEL stores this value inside the **tab-width** user option variable for rst-mode buffers. See Σ **Indentation**. **pel-rst-use-tabs**: whether hard tabs are used for indentation. Defaults to nil (use space characters for all indentation). Σ **Speedbar Support:** • PEL activates Σ **Speedbar** support for reStructuredText when the **pel-use-speedbar** user-option is turned on (set to **t**). Use the Speedbar to see the sections of the reStructuredText document and navigate to them. • **reStucturedText @ Wikipedia** • **Emacs Support for reSTructuredText** • **Basic Intro to rst** • **reStructuredText markup** • **reStructuredText Directives** • **Quick reference to rst** • **rst-cheatsheet (pdf)** • **Docutils** • **Docutils Front-end tools** • **Sphinx @ Wikipedia**, **Sphinx home** • **Sphinx & rst syntax guide** 2026-05-03		
Open this PDF file. See also: Σ **Help/Info**	**<f11> SPC M-r <f1>** **<f12> <f1>**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **M reStructuredText** 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 reStructuredText support	**<f11> SPC M-r <f2>** **<f12> <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL reStructuredText support: pel-pkg-for-rst • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
Σ **Customize** Emacs reStructuredText support	**<f11> SPC M-r <f3>** **<f12> <f3>**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs reStructuredText support: rst • If OTHER-WINDOW is non-nil (use **C-u**), display in another window.
rst-mode	Emacs provides the rst-mode from the rst.el file. The following file extensions are associated with this major mode: `.rst`, `.rest`. PEL adds the `.stxt` extension.		
Activate reStructuredText mode	**M-x rst-mode**	(**rst-mode**)	Toggle the rst-mode used to edit reStructuredText markup. • Automatically invoked when visiting .rst, .rest files (and .stxt files with PEL).
Get version of rst-mode	**C-h v rst-version**		Shows the content of the variable rst-version. Only works once the rst-mode is loaded.
Syntax Control • **superword-mode**	reStructuredText files often benefit from the **superword-mode** and treating underscores as part of words. This helps searching for symbols and opening file that have names that include underscore characters. To force activation of various modes in reStructuredText: first open PEL customization buffer for reStructuredText with **<f12> <f2>**, then • To force **superword-mode** for reStructuredText: add the **superword-mode** to the **pel-rst-activates-minor-modes** user-option. • To force treating underscore as symbol during superword-mode: add **pel-rst-set-underscore-syntax** to the **after** superword-mode in the **pel-rst-activates-minor-modes** user-option list.		

Control syntax of the underscore character
• Requires superword-mode on.
See: Σ **Text Modes**
	<f12> _	(**pel-rst-set-underscore-syntax** &optional ACTION)	Control syntax of underscore to punctuation or symbol when the superword-mode is active.
			• If **superword-mode** is off the function issues a user error. Otherwise the function operates according to the value of the optional argument:
			  • If the argument is not specified or nil, the function toggles the syntax of the underscore character between punctuation (the default) and symbol.
			  • If the argument is positive, it sets the syntax of underscore to symbol.
			  • If the argument is 0 or negative it sets the syntax of the underscore back to punctuation.

Toggle superword-mode See also: Σ **Text Modes**	• **<f11> t m p** • **<f12> M-p** • **M-<f12> M-p**	(**superword-mode** &optional ARG)	Toggle superword-mode: a minor mode that treats snake_case as one word. With prefix argument ARG, enable Superword mode if ARG is positive, disable it otherwise. Useful when writing about code written in Lisp, C, C++, Erlang, Python, languages using the snake_case style.
Editing Content	The following generic commands are useful when editing reStructuredText content.		
• **Filling Text** • Σ **Filling/Justification**	Although text filling will be handled for the generated rendering, you may decide to fill the reStructuredText file itself, after all you're using a markup that's made to allow reading the original text. You can turn the auto fill mode on and identify the fill column. Force the **auto-fill-mode** when a reStructuredText file is visited by adding the **auto-fill-mode** to the **pel-rst-activates-minor-modes** user-option.		
Toggle auto-fill mode	• **<f11> t f C** • **<f11> RET**	(**auto-fill-mode** &optional ARG)	Toggle automatic line breaking (Auto Fill mode). • With a prefix argument, enable Auto Fill mode if the prefix argument is positive, and disable it otherwise. • When Auto Fill mode is enabled, inserting a space at a column beyond 'current-fill-column' automatically breaks the line at a previous space.
Set Fill Column	• **C-x f** • **<f11> t f c**	(**set-fill-column** ARG)	When no prefix value: prompts for column unless a prefix argument was used. • If with **C-u** prefix: use current column. • If with prefix value: use that value.
Fill current paragraph	• **M-q** • **<f11> t f p**	(**fill-paragraph** &optional JUSTIFY REGION)	To justify as well: **C-u M-q** • In refill mode this is done automatically. In auto fill mode the filling is done at the end of the line.
Align a set of lines on some text	**<f11> t a r**	(**align-regexp** BEG END REGEXP &optional GROUP SPACING REPEAT)	Align the current region using an ad-hoc rule read from the minibuffer. BEG and END mark the limits of the region. Interactively, this function prompts for the regular expression REGEXP to align with. • First select a region, then issue the command. For example, to align assignment of variables over the equal sign use **=** as the *regexp*. • The PEL package creates the **ar** alias for **align-regexp**, so it's also possible to invoke it with **M-x ar RET** 💡 Use it to align hyperlink references URL: select all hyperlink lines and then issue the command, specifying **http** as the regexp to line them vertically.
Text Emphasis	The PEL commands emphasize the current word or marked region, then move point to the character right after the emphasized text or inside if empty.		
• **Bold**	**<f12> b** **<f11> SPC M-r b**	(**pel-rst-bold**)	Mark current word or marked region bold. If point after word, use previous word. • Leave point after to the next character. • Inserts required escaped spaces when the emphasized region is inside a word.
• **Italic**	**<f12> i** **<f11> SPC M-r i**	(**pel-rst-italic**)	Mark current word or marked region italic. If point after word, use previous word. • Leave point after to the next character. • Inserts required escaped spaces when the emphasized region is inside a word.
• **Literal**	**<f12> l** **<f11> SPC M-r l**	(**pel-rst-literal**)	Mark current word or marked region with the literal markup. If after word, use previous word. • Leave point after to the next character. • Inserts required escaped spaces when the emphasized region is inside a word.
• **Interpreted**	**<f12> `** **<f11> SPC M-r `**	(**pel-rst-interpreted**)	Mark current word or marked region with the interpreted markup. • Leave point after to the next character. • Inserts required escaped spaces when the emphasized region is inside a word.
Indent list item See also: Σ **Indentation**	**<tab>**	(**indent-for-tab-command** &optional ARG)	With point anywhere on a list item line (a line that starts with one if the supported bullet characters), this cycles the indentation through the possible indentations of the item.
Insert, realign, comment/ uncomment region See also: Σ **Comments** **With PEL**: Comment the current line with **M-0 M-;**	**M-;**	(**comment-dwim** ARG)	Insert or realign comment on current line (or region if a region is active). • On a single line, the comment is placed *after* the code. • **C-u M-;** executes comment-kill ⚠ For reStructuredText, uncommenting does not work.
		(**pel-comment-dwim** ARG)	Same as comment-swim but comments the current line with a numeric ARG or 0.

Description	Keystroke	Function	Note
Itemize all previous lines same indention level ☝	• **<f12> M--** • **M-<f12> M--**	(**pel-itemize-lines** &optional ITEM-PREFIX-STRING)	• Prepend each of the previous lines with a ITEM-PREFIX-STRING (“- “ by default). • Indents all lines above current line that are at the same indentation level. • Use it to put a “- “ prefix on each line instead of typing manually. Put point at empty line after list. Type the command to itemize all lines above. No need to mark.
Duplicate current table underlining This is not a perfect helper for creating reStructuredText table but it helps creating simple ones. For example, to create the top and bottom line from the line under the title:	• **<f12> M-t** • **M-<f12> M-t**	(**pel-rst-table-dup-separator-lines** &optional UPDATE)	Duplicate the current table underlining separator, adding it to the top and the bottom of the table. This command helps you creating a reStructuredText simple table layout, assuming you have only 1 title row and point is at the written separator line. • With the **C-u** option updates the top and bottom line to the line at point.

	Assuming you have the following text and point is on the second line (in blue)	`Header 1  Header 2  Header 3`
		`==========  ==========  ==========`
		`abcdef.  ghijkl.  12345`	

	After executing the command, the top and bottom lines are inserted. The point does not move.	`==========  ==========  ==========`
		`Header 1  Header 2  Header 3`
		`==========  ==========  ==========`
		`abcdef.  ghijkl.  12345`
		`==========  ==========  ==========`	

File’s Table of Content
 	
See also: **Speedbar**
	• Use the **contents markup directive** to generate a table of contents for your reStructuredText file based on its sections.
	• Insert the table of content text inside the file with the **rst-toc-insert** command (although you may want to use the **contents markup directive** instead).
	• Generate a table of content in a buffer to view the file sections and navigate inside them:
	  • with **C-c C-t C-t** to invoke the **rst-doc** command: it opens a *table of Content* buffer, moves point inside it, move to the section title, hit **RET** to select that section inside the original reStructuredText buffer.
	  • using the Speedbar to open a buffer that lists the sections. See **Speedbar**.		

Insert a table content at point ☝ Alternative: use the **contents markup directive**	**C-c C-t TAB**	(**rst-toc-insert** &optional MAX-LEVEL)	Insert the table of contents of the current section at the current column. • By default the top level is ignored if there is only one.
	• A numeric prefix argument MAX-LEVEL overrides ‘rst-toc-insert-max-level’. Text in the line beyond column is deleted.		
Display table of content	**C-c C-t C-t**	(**rst-doc**)	Display a table of contents for current buffer inside the *Table of Contents* buffer. • Displays all section titles found in the current buffer in a hierarchical list.
• **Navigate to specific section**	• Select the section of interest in the *Table of Contents* buffer by navigating to it, then hit **RET** on that section to that section in the file.		
Moving across sections	You can also use the following commands to move to the next or previous section.		
Move to previous section title	• **C-M-a** • **<f12> p** • **<f12> <up>**	(**rst-backward-section** OFFSET)	Jump backward OFFSET section titles ending up at the start of the title line. • OFFSET defaults to 1 and may be negative to move backward. • An OFFSET of 0 does not move unless point is inside a title. • Go to end or beginning of buffer if no more section titles in the desired direction.
	• **<f11> SPC M-r p** • **<f11> SPC M-r <up>**		
Move to next section title	• **C-M-e** • **<f12> n** • **<f12> <down>**	(**rst-forward-section** OFFSET)	Jump forward OFFSET section titles ending up at the start of the title line. • OFFSET defaults to 1 and may be negative to move backward. • An OFFSET of 0 does not move unless point is inside a title. • Go to end or beginning of buffer if no more section titles in the desired • direction.
	• **<f11> SPC M-r n** • **<f11> SPC M-r <down>**		
Mark complete current section	**C-M-h**	(**rst-mark-section** &optional COUNT ALLOW-EXTEND)	Select COUNT sections around point. • Mark following sections for positive COUNT or preceding sections for negative COUNT.
Insert elements based on Tempo skeletons for reStructuredText	PEL provides support for flexible text template insertion through the Emacs built-in **tempo skeleton** mechanism. See also: **Inserting Text** • PEL creates key bindings to invoke the skeletons in the supported major modes, using the same key prefix sequence for each mode: **<f12> <f12>**, with the same key bindings for equivalent concepts (such as file header block) as much as possible. ☝ See also: **Inserting Text** for more info and information about tempo skeleton and yasnippet template-based text insertion).		
Customize PEL Text Insertions control for reStructuredText skeletons.	**<f6> <f2>**	(**pel-customize-pel** &optional OTHER-WINDOW)	Open the customization that control some of the features of the reStructuredText tempo skeletons text inserted by the skeleton text insertion commands for reStructuredText. • If OTHER-WINDOW is non-nil (use **C-u**), display in other window.
	<f12> <f12> <f2>	(**pel-customize-generic-skels** &optional OTHER-WINDOW)	
Insert a file header	**<f12> <f12> h**	(**pel-rst-large-header**)	Insert a large header includes all normal header fields plus separators.
	• Prompts for title. Insert title, updated timestamp, attributes for home page & license, markup for table of contents using the tempo skeleton mechanism. • Automatically activates the PEL tempo skeleton mode so you can move to the target points where extra text must be entered to complete the template.		
Toggle pel-tempo-mode See also: **Mode Line**	**<f12> <f12> SPC**	(**pel-tempo-mode** &optional ARG)	Toggle **pel-tempo-mode** on/off. That mode activates key bindings to move to pre-defined hot-spots where template text must be added.
	The keys are: **C-c .** and. **C-c ,** as well as (in graphics mode only): **C-c C-.** and **C-c C-,** When pel-tempo-mode is active the **pel-tempo-mode lighter** (‡) is shown on the status bar. ☝ When a skeleton is inserted via the execution of one of the pel-rst-… commands, the **pel-tempo-mode** is automatically activated.		
Jump to next tempo mark	• **C-c M-f** • **C-c .** • **C-c C-.**	(**tempo-forward-mark**)	Jump to the next mark in ‘tempo-back-mark-list’: the location where code must be updated inside the inserted skeleton. • These key key bindings are only available when pel-tempo-mode is active.
Jump to previous tempo mark	• **C-c M-b** • **C-c ,** • **C-c C-,**	(**tempo-backward-mark**)	Jump to the previous mark in ‘tempo-back-mark-list’: the location where code must be updated inside the inserted skeleton. • These key binding are only available when pel-tempo-mode is active.
Tempo Template Tag Insertion	**<f12> <f12> <f12>**	(**tempo-complete-tag** &optional SILENT)	Look for a tag and expand it.
	☝ Instead of using the **<f12> <f12>** key bindings above, you can type the template name (shown in the title column like “if”, “case”, etc) completely or partially and then hit **<f12> <f12> <f12>**. A completion buffer opens up if the template name is incomplete (or empty in which case the buffer lists **all** available template names). Select the template name and hit RET. Emacs expands the template. • All the tags in the tag lists in ‘tempo-local-tags’ (this includes ‘tempo-tags’) are searched for a match for the text before the point. The way the string to match for is determined can be altered with the variable ‘tempo-match-finder’. If ‘tempo-match-finder’ returns nil, then the results are the same as no match at all. • If a single match is found, the corresponding template is expanded in place of the matching string. • If a partial completion or no match at all is found, and SILENT is non-nil, the function will give a signal. • If a partial completion is found and ‘tempo-show-completion-buffer’ is non-nil, a buffer containing possible completions is displayed. ⚠ Since, at the moment, only one template is available in rst-mode, the usefulness of this command is limited for reStructuredText.		

Description	Keystroke	Function	Note
Select Section Title Adornment Styles • Adornment styles is the way section tiles are identified: whether the title line has a line under it, another line over it, which character is used and whether the lines are indented.	🎛 The rst.el provides the **rst-preferred-adornment** user-option to select the adornment style of section levels. • You can change this default, add or delete levels and change the character used for the underlining of each section and its indentation. 🎛 The **pel-rst-adornment-style** user-option selects the adornment style: default, Sphinx-Python and CRiSPer. • default supports, by default, a title and 7 section levels; it corresponds to the values selected by **rst-preferred-adornment**. • Sphix-Python supports the 6 levels and formats required by the Sphinx documentation formatter system. • CRiSPer supports a title and 12 section levels. This is what PEL uses by default. • You can of course change the user-option value ay customization, using another default that will persist across Emacs sessions. • PEL also provides 3 commands that you can use to dynamically change the adornment style for the current buffer or globally for all new buffer opened. PEL provides the following commands to show the style used in the buffer and use a different style in then buffer or globally (all buffers opened later in the current editing session). Their key bindings are under the **<f12> <f4>** key prefix. They are also available under **<f11> SPC M-r <f4>** .		
Show current adornment style	**<f12> <f4> a ?**	(**pel-rst-adornment-style-info**)	Display current adornment style in the mini buffer.
Select default adornment style	**<f12> <f4> a d**	(**pel-rst-adorn-default** &optional GLOBALLY)	Set the default section adornment style. With **C-u** prefix: set it globally for the session. This is Emacs rst-mode default: a title with 7 levels.
Select Sphinx-Python adornment style • **Sphinx @ Wikipedia** • **Sphinx home**	**<f12> <f4> a s**	(**pel-rst-adorn-Sphinx-Python** &optional GLOBALLY)	Set the Sphinx-Python section adornment style. With **C-u** prefix: set it globally for the session. This is what Sphinx supports: 6 levels: • parts, • chapters, • sections, • subsections, • subsubsections, • paragraphs.
Select CRiSPer adornment style	**<f12> <f4> a c**	(**pel-rst-adorn-CRiSPer** &optional GLOBALLY)	Set the CRiSPer section adornment style. A title level with another 12 levels. Use **<f12> +** to create those levels.
Section Title level adornment	The rst.el library provides the **rst-adjust** command to create section adornment of the current line, but it is not reliable. PEL provides commands that adorn or change the adornment of the current line to a set of levels according to the selected style.		
Adorn line at title level 💡 Use this at the top of the file.	**<f12> t**	(**pel-rst-adorn-title**)	Adorn current line with level-0 (title) reStructuredText section adornment. • If done at the top of the file, the first adorn line is placed on the first line of the file, a mark is left at the end of the title line and point is moved 2 lines below. • To return to the end of the title line, type **M-`** or **<f6><f6>**.
Adorn to specific level **From level 1 to level 10** 💡 Use this to create a section of a specific level	• **<f12> 1** … • **<f12> 9** • **<f12> 0**	• (**pel-rst-adorn-1**) • (**pel-rst-adorn-2**) ..... • (**pel-rst-adorn-9**) • (**pel-rst-adorn-0**)	Adorn current line with level [1 to 10] reStructuredText section adornment. The **<f12>** keys can only be used in inside the buffers in rst-mode.
Adorn current line: same section level as previous section	• **<f12> =**	(**pel-rst-adorn-same-level**)	Adorn current line with the same level as the previous section. • If the line is already adorned, update the adornment: adjust to previous section level.
Adorn to higher section level	**<f12> +**	(**pel-rst-adorn-increase-level**)	Adorn current line at a higher-level that current if already adorned. • If the line is not already adorned, adorn it with a level higher than previous section.
Adorn to lower section level	**<f12> -**	(**pel-rst-adorn-decrease-level**)	Adorn current line at a lower-level than current if already adorned. • If the line not already adorned, adorn it with a level lower than previous section.
Refresh current line adornment	**<f12> r**	(**pel-rst-adorn-refresh**)	Refresh the adornment of the current line, adjusting the underlining to the current length of the line. Use it when changing the title text.
Adjust section level ⚠ not reliable	• **C-=** • **C-c C-=** • **C-c C-a C-a**	(**rst-adjust** PFXARG)	Auto-adjust the adornment around point. • Adjust/rotate the section adornment for the section title around point or promote/ demote the adornments inside the region, depending on whether the region is active. This function is meant to be invoked possibly multiple times, and can vary its behavior with a positive PFXARG (toggle style), or with a negative PFXARG (alternate behavior). • This function is a bit of a swiss knife. It is meant to adjust the adornments of a section title in reStructuredText. It tries to deal with all the possible cases gracefully and to do “the right thing" in all cases.
Creating and Using Hyperlinks	The following 3 PEL commands help write hyperlink of various forms: • the embedded form where the URL is stored inside the text between angle brackets and • the full named format where the link is located elsewhere in the file on its own line. When editing a buffer using the rst-mode, type the **<f12> .** keystroke to create a hyperlink. • It uses the selected region if one is highlighted or the word at point otherwise as the title for the link and creates the link entry on a line identified by a dedicated bookmark: that bookmark is created by the **<f12> s** keystroke. That helps identify an area inside the file where the next (or several) hyperlinks will be located. • With PEL, the **<f12>** key prefix is mode sensitive. If you want to use the same commands inside another mode, you can use the longer key chord that uses the **<f11> SPC M-r** prefix (assuming that **pel-use-rst** user-option is set).		
Set location of hyperlinks	**<f12> s** **<f11> SPC M-r s**	(**pel-rst-set-ref-bookmark**)	• Set the reference bookmark for the currently edited file at point. • Used to identify the location where the next invocation of M-x pel-rst-mekelink inserts fully expanded links. • Ensures the bookmark is at the beginning of an empty line which is followed by another empty line, by inserting 2 lines and placing the point at the beginning of the first of the 2 lines.
Add an hyperlink for text at point ☝ It's better to use the enclosing syntax (<word>_) as it allows navigation to referenced text with **pel-open-at-point** (**M-<f6>**). Therefore you keep the user-option set to nil (the default).	**<f12> .** **<f11> SPC M-r .**	(**pel-rst-makelink** &optional ARG)	Create a reStructuredText hyperlink prefix for the word at point or region’s text. • If region active, use text of the region for the link, otherwise use the word at point. • If an argument (which can be a **C-u**) is specified, use the embedded URI format. • If no argument is specified, use the named hyperlink format: • if the region is a single word, and **pel-rst-use-single-underscore-for-single-word-ref** user-option is t, just append an underscore to make the link, • if the region is several words, surround it with the "`" and the "`_" strings. • The named link is placed in the location of bookmark named "RST" if it exists and points to same file, otherwise the link is placed at the beginning of the next empty line. • The cursor is placed where the URL is to be written. • Command pushes the mark on mark ring, type **M-`** or **<f6><f6>** to move back to previous location.
Go to hyperlink location	**<f12> g** **<f11> SPC M-r g**	(**pel-rst-goto-ref-bookmark**)	Move point to the reference bookmark. • Useful to see where the bookmark for storing the hyperlink are currently located or add empty lines for future references. • Command pushes the mark on mark ring, type **M-`** or **<f6><f6>** to move back to previous location.
Activating URLs to browse and open files See also: • **Σ File mngt** • **Σ Navigation**	Emacs provides the **goto-url-mode** and the **goto-url-prog-mode** that turn URLs found in the current buffer into clickable buttons. • Once the mode is active the following key sequences are available wheel point is over a URL button: • **C-c RET** or the mouse to click on the button. • If the URL is an email address a buffer to write an email to that address opens. • If the URL is a web or FTP address the system browser is invoked to open the address. • **C-c C-n** : move point to the end of the next URL in the buffer. • **C-c C-p** : move point to to the previous URL in the buffer. • **C-c C-f** : download the file identified by the URL into a local temporary file and visit the file. See (pel-open-url-at-point) above. 🎛 Customization group: **goto-address** . Mostly control the regex for URL and the face used.		
Toggle goto-address-mode	**<f11> f u**	(**goto-address-mode** &optional ARG)	Minor mode to buttonize URLs and e-mail addresses in the current buffer. With a prefix argument ARG, enable the mode if ARG is positive, and disable it otherwise.
Toggle goto-addrress-prog-mode	**<f11> f U**	(**goto-address-prog-mode** &optional ARG)	Like ‘goto-address-mode’, but only for comments and strings.

Description	Keystroke	Function	Note

Open the URL (email or web page)	`C-c RET`	(**goto-address-at-point** &optional EVENT)	Open the URL at point:
			• If URL is a web page: open it in a browser
			• If URL is a mail address:
			  • Send mail to address at point:
			  Find e-mail address around or before point. Then search backwards to beginning of line for the start of an e-mail address.
			  • If no email address is found there, then load the URL at or before point.

Move to end of next URL in buffer See also: **Navigation**	`C-c C-n` `<f6> C-n`	(**pel-goto-next-url**)	Move point forward to the end of the next URL located in the current buffer. • The global `<f6> C-n` key binding activates the goto-address-mode if it is not already active.
Move to beginning of previous URL in buffer See also: **Navigation**	`C-c C-p` `<f11> C-p`	(**pel-goto-previous-url**)	Move point backward to the beginning of the previous URL located in the current buffer. • The global `<f6> C-p` key binding activates the goto-address-mode if it is not already active.
Copy URL at point in temporary file and visit the file See also: **File mngt**	`<f11> f M-u` `C-c C-f`	(**pel-open-url-at-point**)	Copy the URL at point to a local temporary file and visit that file. • ⚠️The download copy of the file does not have the same name and may not open with the proper mode because it won't have an extension. The HTML formatted files will be recognized by Emacs but most of the files won't be. • Save the file somewhere else using the **C-x C-w** key sequence and identify the proper extension to activate the required major mode. ☝ This binding is only available when point is over the URL and the **goto-address-mode** minor mode is active. Use **<f11> f u** or **<f11> f U** to activate this mode.

Open file or web-page whose name is at point
 	
★★
Command is generic and is also specialized for:
• **C**
• **C++**
• **Erlang**
• **UNIX Shell**
 	
Jump to referenced link (unless N >= 100) ☛
 	
Delimiting characters ☛
 	
File identification heuristic ☛
`<f11> f <f2>` ☛
`<f11> f ;` ☛
 	
Select multi-file selection method ☛
 	
Select target window ☛
 	
N>20 : open the directory ☛
 	
See function docstring for more info.
	• `M-*`
	• `<f11> f .`
	• `6y`
		(**pel-open-at-point** &optional N)	Open the file, library or the URL, named at point, with potential line & column #s.
			🎛 Supports glob characters, partial directory path. When multiple files are found it prompts using the method selected by **pel-prompt-read-method** user-option.
			📦 The **6y** key-chord is available if **pel-use-key-chord** is non-nil. See **Key-Chords**.
			
			☝ This command works generically but is also specialized for reStructuredText: **Inside a rst-mode** buffer, when the point is over a **reStructuredText external hyperlink target** *reference*, the command locates the reference and opens the file identified by the reference (*unless N >= 100*):
			• If the reference is a web URL, it opens the identified web using the system browser.
			• If the reference is a HTML file name that corresponds to the rendering of a local restructuredText file, it opens that reStructuredText file.
			• If the reference is a HTML file that does not correspond to a reStructuredText source, it opens that HTML file.
			• If the reference is another type of file it opens that file.
			• If the reference URL identifies a # **URI fragment** that identifies the name of a target file section, the command moves point to the section
			• If the hyperlink refers to a title inside the document **move point to that title** instead of trying to open a file.
			
			In general the command extracts the file or directory name, and possibly line and column numbers, from text at point and tries to open the file or directory.
			• The generic mode extraction works by identifying the beginning & end of the file/directory/library/URL name string by delimiter characters, one of: tab, newline and: **“ ` ’ | ( ) [ ] { } <>** ’ ’ “ ” 「 」 〔 〕 〈 〉 《 》 【 】 〖 〗 « » ‹ › ❮ ❯ ❬ ❭ ❰ ❱ ❲ ❳ ❨ ❩ ❪ ❫ ❴ ❵ ❛ ❜ ❝ ❞ ❟ ❠ ⟦ ⟧ ⟨ ⟩ ⟪ ⟫ ⟮ ⟯ ⦃ ⦄ ⦅ ⦆ ⦇ ⦈ ⦉ ⦊ ⦋ ⦌ ⦍ ⦎ ⦏ ⦐ ⧘ ⧙ ⧚ ⧛ ⸢ ⸣ ⸤ ⸥ ⸦ ⸧ ⸨ ⸩ 〈 〉 《 》 「 」 『 』 【 】 〔 〕 〖 〗 〘 〙 〚 〛 ﴾ ﴿ ｢ ｣ ・。
			• If embedded space(s) are allowed in the name, point must be located at the first of the 2 delimiter chars. Otherwise point can be anywhere in the name.
			• The name may include glob characters
			
			The command uses a URL unchanged but uses the following heuristic to identify the exact location of the file/directory:
			• In the file/dir name is an absolute path it uses that. Otherwise
			• it builds a absolute path using the extracted relative path name inside the directory identified by the **pel-open-file-at-point-dir-home** user-option, which can be 1) use parent directory of currently visited file, or use current working directory, 2) use current working directory, or 3) use user-specified directory. It uses the found file/dir name if it exists. Otherwise
			• it searches for the relative file/dir name in directory tree under the root marker file identified by the **pel-project-root-identifiers** user-option which is something like .git, .hg, .project, .pel-project (the default). If it can find such a file in the above directories it searches the tree under the found root.
			• If it finds several files it prompts using the current completion mode to allow selection of the appropriate name (see below) and opens the selected one.
			• If it finds only one it opens that file. Otherwise,
			• it prompts showing the name searched and provide the following choices: 1) create the file with specified name, 2) edit the name to search again, 3) use the name found and search for an Emacs library file with that name, or 4) quit.
			
			The command opens the extracted name according to this heuristic:
			• If the string is a properly formatted **URL**, it opens it using the OS default browser (even if a optional numeric argument specified otherwise), otherwise
			• if the string is a file or directory name it opens it.
			• If the file name is followed by **line and column numbers** the point is moved to that position in the buffer.
			
			🎛 When finding several file names, the command lists them and prompts using the method selected by **pel-prompt-read-method** user-option.
			• The default is a very primitive function implemented by PEL. You can select a more powerful **ivy** prompting instead.
			  • With **ivy** selected, PEL will automatically set 📦 **pel-use-ivy** to **t** 📦 and **Ivy mode** will be installed automatically when you restart Emacs.
			• Note that the command shows all files found by the specified search method, it does not only use the first one found.
			  • ☝ Use this to detect potential duplication in header file names in large include paths.
			
			The command opens the file in the window selected by the following logic controlled by presence or absence of typed numerical prefix arguments:
			• **Select target window:**
			  • Without argument:
			  • If file or directory is already opened in a window, move point to that window and to the line column coordinates if specified.
			  • If no window holds that file, select the target window according to the number of editable windows in frame: if 1, split that window and use the new window, if 2: use the other window, if 3 or more, use the current window.
			  • With prefix numeric argument N:
			  • N < 0 : create a new window and use that.
			  • (abs N) > 20: then open the **directory** instead of the file. Interpret the window position from the N value adjusted: N-20 (or N+20 if N is negative)
			  • N = 0: use the '*other*' (the next) window.
			  • N = 1, 3, 7or above (excluding 8, 9 and 10): select the target window based on the number of editable windows in frame:
			  • if 1 window: split that window and use the new window,
			  • if 2 windows: use the other window,
			  • if 3 or more windows: use the current window.
			  • N is: 8: up, 2: down, 4:left, 5:current, 6:right.
			  • N is 9: force **opening the file in the OS associated application** (with N=29 or N=-29, open the file's directory with the OS associated application (eg. macOS Finder, Windows Explorer). If this is a URL, open it in the OS default web browser.
			  • ☝ If N >= 100, restructured hyperlink referenced is ignored, text is used as the file target and N-100 is used to determine the window selection.
			  • Selecting Minibuffer, inexistent or dedicated window is not allowed.

Compile/render			Compiling a reStructuredText file into any of the supported format using a command identified by the **pel-rst-compiler** user-option. • The default value of pel-rst-compiler generates a HTML file from the restructuredText using the **Docutils front-end tool**: **rst2html**.
Compile the file: generate rendered file See also: **Compilation Mode**	`<f12> c`	(**pel-rst-compile**)	"Compile" the reStructuredText file into the rendered format (html or something else). • The compiler command is selected by the **pel-rst-compiler** user-option. It defaults to *pel-rst2html*, *which* is treated specially: if it is not found in PATH, PEL automatically adjusts the command line to use the script provided in its bin directory. That script uses **rst2html** to generate an html file. • You can specify any other command line with or without arguments, as long as the name of the processed reStructuredText file is the last argument of that command (because PEL appends the name of that file to the command). • All errors are shown in a *compilation* buffer. See **Compilation Mode** for more info.
Move point to next compilation/ rendering error	• `C-x ` • `M-g n` • `M-g M-n`	(**next-error** &optional ARG RESET)	Move to the next error message and visit the corresponding source code. • If all the error messages parsed so far have been processed already, the message buffer is checked for new ones. • A prefix ARG specifies how many error messages to move; negative means move back to previous error messages. • Just `C-u` as a prefix means reparse the error message buffer and start at the first error.

Description	Keystroke	Function	Note
Move point to previous compilation/rendering error	• `M-g p` • `M-g M-p`	(**previous-erro**r &optional N)	Visit previous previous error message and corresponding source code. • Prefix arg N says how many error messages to move backwards (or forwards, if negative).

rst-mode — References

Description & URL	Notes
Emacs Support for reStructuredText	
reStructuredText	Main page for all reStructuredText documents.
reStructuredText markup Specifications	Formal markup specifications.
Sphinx Python Documentation Generator	• **Sphinx — Documentation Contents** • **Sphinx — Documentation —Sections**