Tags-based Cross Reference Creation and Navigation 🚧

Description	Keystroke	Function	Note

Finding, Replacing and Managing Identifier References	Emacs provides a cross reference mechanism that supports multiple backends tied to the major mode of the current buffer.
	• One of these back-end is the Xref-Etags mode. That mode uses an external TAGS file with the etags syntax supported by the etags utility and other etags-compatible tools (such as the Universal Ctags).
	• Other backend exist, depending of the file type and the associated major mode. For example, Emacs has a native backend for Emacs Lisp files that uses the information from the loaded Emacs Lisp files which it uses by default and does not require a TAGS file.
	• It is possible to activate the Xref-Etags mode to add ability to locate identifiers from the information provided by the TAGS file.
	  • See information on how to create the TAGS file.
	
	This table contains:
	• **Customization Groups :** Information about the Customization Groups affecting cross referencing.
	• **Xref-ETags Mode :** How to control use of the Xref-Etags mode
	• Commands that require TAGS files and Xref-Mode:
	  • **Searching/Replacing via TAGS file**
	  • **Inquiries with TAGS file**
	• Commands that can use TAGS file with Xref-Mode but also other backends:
	  • **Looking Up Identifiers**
	  • **Identifier Inquiries**
	  • **Searching/Replacing Identifiers**
	• **Operations in the *xref* buffer**
	• **Creating TAGS files - Examples**		

Customization Groups 🚧	The following customization groups are used to manage the user options that affect the cross reference mechanism identified in that table.		
Group: xref, etags			
Group: Speedbar (tag browser)			
Group: Helm Tags			
Group: Pel Tags			
Xref-Etags Mode	The following helper commands can be used to toggle the Xref-Etags mode and display its current status.		
Toggle the Xref-Etags mode on/off	`<f11> X X`	(**xref-etags-mode** &optional ARG)	Toggle etags-based search mode on/off. • Certain major modes install their own mechanisms for listing identifiers and navigation. Turn this on to undo those settings and just use etags.
Display state of the Xref-Etags mode (See also ∑ Help/Info)	• `<f11> X ?` • `<f11> ? X`	(**pel-show-etags-mode-status**)	Display current state of tags-based search mode and active etags TAGS file. It displays: • the state of the xref-etags-mode variable for the current buffer, • the current value of the tags-file-name variable • the active value of the **tags-table-list** user option. ⚠️ Do not change tags-file-name variable manually. Use visit-tags-table (**<f11> X T**) to modify it.
Searching/Replacing via TAGS file	The following commands perform search and replace operations that are always based on the information found inside a TAGS file (created by the etags utility or something compatible). • The TAGS file currently used is stored inside the tags-file-name variable. user option but also set via the directory locals variable of the same name stored inside the .dir-locals.el file in the current directory or a directory above. • PEL provides binding for the commands that have no binding by default. ☝ The following commands require a valid TAGS file created by the etags or an etags-compatible tool.		
Select the TAGS file for TAGS-based search/replace operations	`<f11> X T`	(**visit-tags-table** FILE &optional LOCAL)	Tell tags commands to use tags table file FILE. Propose TAGS file in current directory. • FILE should be the name of a file created with the 'etags' program. • A directory name is ok too; it means file TAGS in that directory. • Normally M-x visit-tags-table sets the global value of 'tags-file-name'. • With a prefix arg, set the buffer-local value instead. ☝This sets the variable tags-file-name. But it also prompts to update the value of the **tags-table-list** user option which may contain the path/name of several TAGS files. If you work with files in various projects stored in different directory trees, you may store TAGS file at the root of each of these directory and use this command to use that TAGS file when you need it.

Search for identified in the TAGS file	`<f11> X S`	(**tags-search** REGEXP &optional FILE-LIST-FORM)	Search through all files listed in tags table for match for REGEXP.
			• Stops when a match is found.
			• To continue searching for next match, use command M-x tags-loop-continue.
			• The search is done in the current TAGS file.
			  • It is identified by the tags-file-name variable .
			  • It can be customized to select a default.
			  • Values for various projects can be identified in a directory local file (.dir-locals.el) , see the ∑ File/Directory Variables table.
			  • ⚠️ Do not modify tags-file-name manually. Either:
			  • change the global customized value through customization, or
			  • change the directory locals by editing the .dir-locals.el file, or
			  • change the currently active value by executing the **visit-tags-table** command.

Replace regexp via TAGS file	`<f11> X R`	(**tags-query-replace** FROM TO &optional DELIMITED FILE-LIST-FORM)	Prompt for a regexp search string, a replacement string and search though all files listed in the tags table for a match. Prompt for first match found and allow repeat. • With argument prefix (**C-u)** replace only whole words.
Repeat last TAGS-based search/replace	`<f11> X N`	(**tags-loop-continue** &optional FIRST-TIME)	Continue last M-x tags-search or M-x tags-query-replace command. • Two variables control the processing we do on each file: the value of 'tags-loop-scan' is a form to be executed on each file to see if it is interesting (it returns non-nil if so) and 'tags-loop-operate' is a form to evaluate to operate on an interesting file. If the latter evaluates to nil, we exit; otherwise we scan the next file.
Inquiries with TAGS file	The following commands perform operation related to TAGS files and use the tag backend. • The list-tags displays the identifiers defined in a specified file. • The next-file visit files where identifiers are defined, one at a time. ⚠️ These commands only work with the etags backend and require an accessible TAGS file.		
List identifiers defined in a specified source file	`<f11> X L`	(**list-tags** FILE &optional NEXT-MATCH)	Display list of tags in file FILE. • This searches only the first table in the list, and no included tables. ☝The etags file format supports an "include" statement that includes other etags file. Keep that in mind to decide if you want to use that etags feature. • FILE should be as it appeared in the 'etags' command: files that are located in the same directory as the TAGS file do not specify the directory, the source files located in a sub-directory of the directory holding the TAGS file will have one. • The list of all tags for this file are shown inside a *Tags List* buffer opened in apropos-mode: type **<ret>** on a line to move to the definition, **q** to close the window.
Vist files with identifier definions	`<f11> X F`	(**next-file** &optional INITIALIZE NOVISIT)	Select next file among files in current tags table. • A prefix arg initializes to the beginning of the list of files in the tags table.

Description	Keystroke	Function	Note

Looking Up Identifiers
 	
(finding identifier definitions)
	• The first 4 commands find or prompt for identifier or patterns and request the backed to perform the search.
	  • The search backed depends on the major mode. Elisp, for example, uses info from compiler and load path by default.
	  • If the identifier is not found, you can force search for buffer to use a TAGS file created by tags (or equivalent tool) by executing `xref-etag-mode`.
	  • If multiple identifiers are found they are listed inside the *xref* buffer for selection.
	• To move back to the original location use the `xref-pop-marker-stack` command, with the **M-,** key.		

Find definition of identifier at point	**M-.**	(**xref-find-definitions** IDENTIFIER)	Grab symbol at point and move cursor to its definition. • If there are more than one match, prompt in the *xref* buffer. • To search for a symbol entered manually, type **C-u M-.**
Find definition of identifier at point, display in other window	**C-x 4 .**	(**xref-find-definitions-other-window** IDENTIFIER)	Same as **M-.** but opens inside another window.
Find definition of identifier at point, display in other frame	**C-x 5 .**	(**xref-find-definitions-other-frame** IDENTIFIER)	Same as **M-.** but opens inside another frame.
Go back to where M-. was last issued	**M-,**	(**xref-pop-marker-stack**)	• Pop back to where M-. was last invoked. • Marker depth is controlled by the **xref-marker-ring-length** user option.
Identifier Inquiries	The following commands perform other inquiries on the identifiers using the backend search mechanism used for the current buffer.		
Symbol Completion at point (See also: ⨋ Auto-Completion)	• **C-M-i** • **M-<tab>**	(**completion-at-point**)	Perform completion on the text around point. • The completion method is determined by '**completion-at-point-functions**'. • The tags-completion-at-point-function is used for Emacs Lisp code by default. It provides a list of possible values in the *Completions* buffer. ☝ This key binding is also used for Flyspell, which can be used to spell check only moments and strings. See the specific programming language tables for more information.
Find all identifiers that match a regex pattern	• **C-M-.** • **<f11> X .**	(**xref-find-apropos** PATTERN)	Find all meaningful symbols that match PATTERN. • PATTERN is a regex. • The argument has the same meaning as in 'apropos'.
Searching/Replacing Identifiers (finding where identifiers are referenced)	With the commands in this group you can: • locate where a given identifier is used/accessed/defined, (listing them in the *xref* buffer), • replace the identifier names in all location where it was found, or • replace identifiers matching a regexp to a new value in all the locations where they were found. The search/replace operations can be a useful tool in code refactoring.		
List all references to symbol at point	**M-?**	(**xref-find-references** IDENTIFIER)	Grab the symbol at point, maybe prompt (with input completion) and find all references for identifier and display them in the *xref* buffer window. • Backend determines if prompting is done. Some backbends prompt even if point is at valid identifier. • To force always prompting set the **xref-prompt-for-identifier** user option to **t**.
Interactively replace identifier in current and next references.	**<f11> X r**	(**xref-query-replace-in-results** FROM TO)	Interactively replace current identifier in current and next references with another string. • Prompts for the current xref (but you can normally just hit RET to accept it) and the replacement. Then brings the xref noised another window and prompts for the action. Hit **?** for possible actions. ☝ It's best to use this from the *xref* buffer: inside the buffer you just have to type the **r** key. See below.
Move to location of first xref found	**<f11> X 1**	(**first-error** &optional N)	Restart at the first xref found. • Visit corresponding source code. With prefix arg N, visit the source code of the Nth error.
Move to next xref found	• **C-`** • **M-g n** • **M-g M-n**	(**next-error** &optional ARG RESET)	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.
Move to previous xref found	• **M-g p** • **M-g M-p**	(**previous-error** &optional N)	Prefix arg N says how many error messages to move backwards (or forwards, if negative).
Operations in the *xref* buffer	The results of an identifier search are displayed in the *xref* buffer. When point is inside this buffer the following operations are available: • jumping to the file/location where the identifier was found and described in the current *xref* buffer line and either moving point into that window or keeping it inside the *xref* buffer window. • Jumping to the next or previous cross reference. • Performing replacement of the identifier in all its cross references. • Navigating through the lines of the *xref* buffer with some extra quick keys, in addition to the normally accessible navigation commands. These commands are shown below. Use the **q** key to close the *xref* buffer window.		
Jump to current xref	**RET**	(**xref-goto-xref** &optional QUIT)	Jump to the xref on the current line and select its window. • If a window is already opened for the file it uses it, otherwise it opens a new window.
	C-o	(**xref-show-location-at-point**)	Display the source of xref at point in the appropriate window, if any. • If a window is already opened for the file it uses it, otherwise it opens a new window.
Jump to current xref, quit *xref* buffer	**<tab>**	(**xref-quit-and-goto-xref**)	Quit *xref* buffer, then jump to xref on current line.
Move to previous xref line and display its source	• **,** • **p**	(**xref-prev-line**)	Move to the previous/next xref and display its source in the appropriate window. • If a window is already opened for the file it uses it, otherwise it opens a new window. • Point stays in the *xref* buffer.
Move to next xref line and display its source	• **.** • **n**	(**xref-next-line**)	
Interactively replace identifier in current and next references.	**r**	(**xref-query-replace-in-results** FROM TO)	Interactively replace current identifier in current and next references with another string. • Prompts for the current xref (but you can normally just hit RET to accept it) and the replacement. Then brings the xref noised another window and prompts for the action. Hit **?** for possible actions.
Scroll buffer up	• **SPC** • **C-v**	(**scroll-up-command** &optional ARG)	Scroll text of selected window upward ARG lines; or near full screen if no ARG.
Scroll buffer down	• **S-SPC** • **DEL (⌫)**	(**scroll-down-command** &optional ARG)	Scroll text of selected window down ARG lines; or near full screen if no ARG.
Move to beginning of buffer	**<**	(**beginning-of-buffer** &optional ARG)	Move point to the beginning of the buffer.
Move to end of buffer	**>**	(**end-of-buffer** &optional ARG)	Move point to the end of the buffer.
Quit the *xref* window	**q**	(**quit-window** &optional KILL WINDOW)	Quit *xref* window and bury its buffer.

Description	Keystroke	Function	Note
Creating TAGS files - Examples	The following commands can be used to create etags-compatible TAGS files. • In the first set you see a set of commands that can be executed manually using the `M-x` and the `M-!` commands to execute specific shell commands. • The etags utility is part of GNU Emacs distribution, normally you should have access to it from your PATH. If not, add its directory to PATH prior to executing these commands.		
Display (and optionally change) current directory	`M-x cd`		Move to the directory that must contain the TAGS file. If you want to create TAGS files that contain relative file paths then you should move to where the files of your project are located.
Display etags help	`M-! etags –help`		Display the help information for the etags command line utility. • The result is shown in the *Shell Command Output* buffer.
Create a etags-compliant TAGS file for Elisp files of current directory	`M-! etags *.el`		Create a TAGS file in the current directory for all its Emacs Lisp files.
Create a etags-compliant TAGS file for Elisp files of 2 directories	`M-! etags *.el other/*.el`		Create a TAGS file in the current directory for all its Emacs Lisp files and all Emacs Lisp files in the sub-directory other.
Create a etags-compliant TAGS file for .py Python files in current directory tree	• `M-! find . -type f -name *.py -print > all.txt` • `M-! etags - < all.txt`		Create a TAGS file in the current directory for all Python source code files located inside the directory and all its sub-directories. • Using 2 commands storing the output of the find command into the file all.txt then passing its content to etags standard input. • Using the shorter pipe for one command does not work with M-xX
Create a etags-compliant TAGS file for .py and .pyw Python files in current directory tree	• `M-! find . -type f \( -name *.py or –name *.pyw \) -print > all.txt` • `M-! etags - < all.txt`		Same as above except that include both the .py and the .pyw files.

References — Tags

Topic & Link	Description
Learning GNU Emacs - Ch 9 - Computer Language Support	
Using CTags	
CTags - wikipedia	Lists various tags processing programs, including the various CTags and Etags (the emacs tags)
CTags - A maintained ctags implementation https://ctags.io	
CTags - Universal-ctags Hacking Guide	Universal Ctags continues the development of the now-defunct Exuberant CTags. Universal CTags is maintained.
Tag Tools pages	
ctags	help available in man page. in /usr/bin : restricted.
etags	Comes with GNU emacs; info available in man page.
ExuberantCTags	According to the EmacsWiki (https://www.emacswiki.org/emacs/ExuberantCtags) this supports more languages than etags. Apparently this project is no longer maintained; Universal CTags is a fork and is maintained.
Universal CTags	Homebrew has a tap for installing Universal CTags: https://github.com/universal-ctags/homebrew-universal-ctags
Hasktags	
Emacs and CTags	
Using CTags	
CTags - wikipedia	Lists various tags processing programs, including the various CTags and Etags (the emacs tags)
CTags - A maintained ctags implementation https://ctags.io	
CTags - Universal-ctags Hacking Guide	Universal Ctags continues the development of the now-defunct Exuberant CTags. Universal CTags is maintained.
CTag Tools	
ctags	help available in man page. in /usr/bin : restricted.
etags	Comes with GNU emacs; info available in man page.
ExuberantCTags	According to the EmacsWiki (https://www.emacswiki.org/emacs/ExuberantCtags) this supports more languages than etags. However, apparently this project is no longer maintained; Universal CTags is a fork and is maintained.
Universal CTags	Homebrew has a tap for installing Universal CTags: https://github.com/universal-ctags/homebrew-universal-ctags
Notes on installing Universal Ctags on a macOS system	On my macOS system, I installed universal ctags which has an executable that is named ctags and placed inside /usr/local/bin (which is before /usr/bin where the original ctags is located. • Homebrew removed the man page for the original ctags. I would have preferred hey used a different name for universal ctags (something like uctags) but they did not do that. The ctags man page is now the page for universal ctags… • Universal ctags has a mode for emacs. Also note that tags was not removed by the installation of Universal ctags. So I manually renamed Universal ctags, which is a symlink in /usr/local/bin to **uctags**, so that I can still access the original ctags if needed. To access the original ctags man page use : “man -a ctags” this will open all ctags man pages one after the other (when one is closed) and after closing the universal ctags page, the original cat page is opened.
Using Tags with Erlang	
Etags with Erlang @ erlang.org	Describes how to use tags with Erlang source code and how to create the TAGS file.
Hasktags	