Cross Reference Creation and Navigation

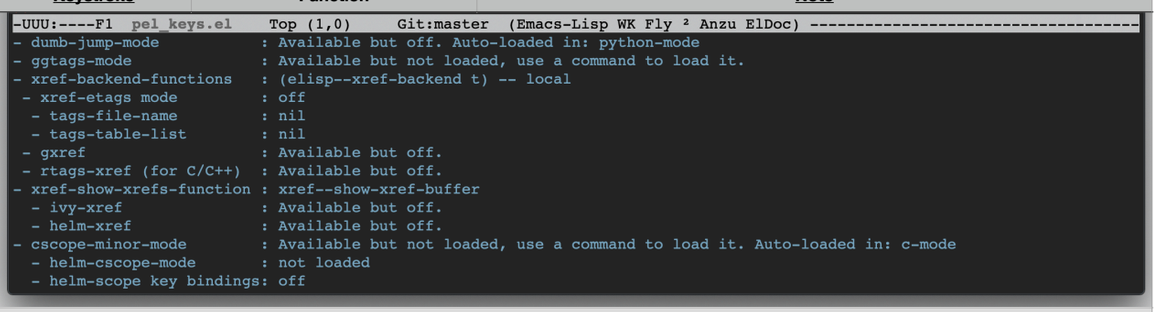
Description	Keystroke	Function	Note

Cross References with Emacs	Emacs provides several cross reference tools. Some tools are unified under the **unified xref interface** some others are independent.
	PEL support several of them:
	• The **unified xref interface** is available since Emacs version 25.1. This includes:
	  • The xref unified interface can be used with various **back-ends**:
	  1. major-mode specific load/interpretation backend (such as what is used for Emacs Lisp)
	  2. Tags-based tools using external TAGS file with the **etags syntax** supported by the etags utility and other etags-compatible tools:
	  • etags (Emacs tags utility) ; see **how to create TAGS files with etags**, used by the **xref-etag-mode**.
	  • **Universal Ctags** (successor of Exuberant Ctags)
	  • **GNU GLOBAL** gtags utility with Universal Ctags and **Pygments** plugin.
	  • The **gxref** xref back-end
	  3. other specialized parsers based tools that do not use tags:
	  • Programming language agnostic packages:
	  • **dumb-jump**, a fast grep/ag/ripgrep-based engine to navigate in over 40 programming languages without tags/database index files.
	  • Specialized packages for specific major modes:
	  • **rtag-xref**, a **RTags** backend specialized for C/C++ code. It uses a client/server application.
	  • **info-xref** an internal package used to navigate into info document external references.
	  • The xref unified interface can also be used with various **front-end** selectors when several entries are found for a search:
	  • The default xref selector that uses a *xref* buffer to show all search result
	  • The **ivy-xref** interface to select candidates with **ivy**
	  • The **helm-xref** interface to select candidates with **helm**.
	• Independent Emacs cross reference packages:
	  • The **ggtags** package that provides direct access to GNU Global gtags-made tags database.
	  • **CScope** command line utility via **xcscope** package (potentially as a font-end for GNU Global)
	
	PEL provides:
	• a set of user options the control the installation and use of the various Emacs packages mentioned above:
	  • **pel-use-xcscope, pel-use-helm-cscope** and **pel-modes-activating-cscope**
	  • **pel-use-dumb-jump**
	  • **pel-use-ggtags**
	  • **pel-use-gxref**
	  • **pel-use-rtags** and **pel-use-rtags-xref** (🚧 support not completed yet)
	  • **pel-use-ivy-xref**
	  • **pel-use-helm-xref**
	• a set of user options to identify major modes that automatically activate a xref backed or a xref front-end:
	  • **pel-modes-activating-dumb-jump**
	  • **pel-modes-activating-gxref**
	  • **pel-modes-activating-ggtags**
	• Command to dynamically control the xref back-end:
	  • **`<f11> X D`** : pel-toggle-dumb-jump-mode
	  • **`<f11> X G`** : pel-toggle-gxref-mode
	• Command to dynamically select the xref front-end: pel-select-xref-front-end : **`<f11> X F`**
	
	For the tags-based tools, the **∑ Projectile** package can be used to create the TAGS file for all project files.
	• The projectile-tags-backend user option allow selection of the tags backend from:
	  • automatic: selects first available in following order: ggtags, xref, etags, find-tag
	  • xref, ggtags, etags or standard selected explicitly.
	
	Aside from help and customization, there are 3 types of commands involved in cross reference management:
	1. the commands that activate/de-activate/toggle cross-reference handling modes
	2. the commands that look up identifiers, moving point to identifier definition, listing references, etc… These commands, or the key binding of these commands are mostly the same for all modes
	3. auxiliary commands that provide extra functionality like grep, query-replace, operate within the project files, build or rebuild the tag databases, etc, all using the cross referencing mechanism used.		

	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 / How to start using the Xref-Etags mode.**
	• **Operations in the *xref* buffer**
	• **Creating TAGS files - Examples**		

Open this PDF file. See also: **∑ Help/Info**	**`<f11> X <f1>`**	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the local copy of the **∑ Xref** PDF file unless a command prefix (like **`C-u`**) was used. In that case it opens the Github-hosted file instead.
Customization Groups	The following customization groups are used to manage the user options that affect the cross reference mechanism identified in that table.		
Activate/Select PEL cross-reference control	**`<f11> X <f2>`**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL cross-reference support: gtags, dumb-jump • If OTHER-WINDOW is non-nil (use **`C-u`**), display in other window.
∑ Customize Emacs cross-reference control	**`<f11> X <f3>`**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs cross-reference support: dumb-jump, etags, ggtags, helm, projectile, speedbar, xref
Activate ∑ Projectile	**`<f11> <f8> <f2>`**	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL cross-reference support: gtags, dumb-jump • If OTHER-WINDOW is non-nil (use **`C-u`**), display in other window.
Customize ∑ Projectile	**`<f11> <f8> <f3>`**	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Projectile. The following user options control the creation of tag file for the entire project: **projectile-tags-command** and **projectile-tags-file-name**.
Active tools info	Emacs supports a large set of tools to perform cross-reference searching. • PEL provides bindings to a large set of tools and command that displays the currently active tools: use the **`<f11> X ?`** key sequence to list them.		
Display state of the Xref back-end and other cross referencing modes **See also: ∑ Help/Info**	• **`<f11> X ?`** • **`<f11> ? X`**	(**pel-show-etags-mode-status**)	Display current state of cross-referencing modes as described below

	It displays:
	• the state of ggtags-mode, the state of dumb-jump
	• the state of xref back-end, with:
	  • the value of **xref-backend-functions**
	  • the state of the **xref-etags-mode** variable for the current buffer, the current value of the **tags-file-name** variable and 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.
	  • the state of gxref
	  • the state of rtags-xref (for C/C++)
	• the state of the xref front-ends:
	  • xref, which displays results in a *xref* buffer
	  • helm-xref, which displays the results inside a helm buffer. This provides a large set of helm-related functionality for further searches and actions.
	  • ivy-ref, which displays the results in a simple ivy list.
	
	See example screenshot below.		

Description	Keystroke	Function	Note
Example of Xref tool status when editing an Emacs Lisp file.			

```
-UUU:----F1  pel_keys.el    Top (1,0)     Git:master  (Emacs-Lisp WK Fly ² Anzu ElDoc) ---------------------------------
- dumb-jump-mode           : Available but off. Auto-loaded in: python-mode
- ggtags-mode              : Available but not loaded, use a command to load it.
- xref-backend-functions   : (elisp--xref-backend t) -- local
 - xref-etags mode         : off
  - tags-file-name         : nil
  - tags-table-list        : nil
 - gxref                   : Available but off.
 - rtags-xref (for C/C++)  : Available but off.
- xref-show-xrefs-function : xref--show-xref-buffer
  - ivy-xref               : Available but off.
  - helm-xref              : Available but off.
- cscope-minor-mode        : Available but not loaded, use a command to load it. Auto-loaded in: c-mode
  - helm-cscope-mode       : not loaded
  - helm-scope key bindings: off
```

Selecting and Using Cross Reference Tools

The tools to use depend on the programming and markup language(s) of the files you want to inspect.
- Some tools support several files types, other are specialized for specific file types.
- Some tools are mutually exclusive. Some tools can be used with other tools at the same time.
  - For example if you navigate Emacs source, you need support for Emacs Lisp and C. The built-in xref system support several back-ends but some of the backends cannot be used with some others. By default the elisp backend is active but it will handle only symbols that are part of already loaded files. Using the etags xref backend requires the creation of an etags-compatible TAGS file but that will allow to navigate to all symbols, including the ones defined in C source code files. With the etags-xref-backend on, you can also activate the cscope-mode for the C source code files, which will provide extra capabilities for the C source code files.

You can select the tools dynamically or from the PEL user option variables, as described in the first cell above.
- However, for best results, you need to activate the modes when a file is opened, via the hooks. PEL provides user option variables to identify these automatic loading of the modes.
  - **pel-modes-activating-cscope**
    - pel-modes-activating-helm-cscope
  - **pel-modes-activating-dumb-jump**
  - **pel-modes-activating-ggtags**
    - pel-modes-activating-gxref

The following commands are available to dynamically select the tools and modes.

• Select CScope

Supports:
- C, C++, Java

CScope is mainly used to index C source code. It also has partial support for C++ and Java.
- Although the CScope project is not very active in 2020, it can still be used to navigate C source code; it provides some features not available elsewhere.
- PEL provides commands you can use to quickly activate or deactivate CScope.
- When CScope mode is active, a Cscope menu entry is available. Use **<f10>** to open it.

📦 This require the **xcscope** external package and the **CScope** command line utility.

PEL activates **xcscope** when **pel-use-xcscope** user option is **t**. You must install the **CScope** command line utility yourself.

Additionally, the helm-cscope-mode provides the ability to view Cscope search results with **helm**.

📦 It requires **helm-cscope** external package PEL enables this command when **pel-use-helm-cscope** is set to **t**.

If you want this mode automatically activated in one of the supposed major modes, add the modes to the **pel-modes-activating-helm-cscope** user option.

Building Cscope database:

☝ To use the cscope mode you must first build a CScope database. PEL provides the **bin/cscope-c** and **bin/cscope-cpp** shell script that lists C and C/C++ files into the cscope.files and then build a Cscope database from the identified source code files.

Toggle CScope interaction with cscope-minor-mode	**<f11> X C C**	(**cscope-minor-mode** &optional ARG)	Toggles the cscope minor mode on/off. • With cscope-minor-mode on, the cscope keybindings are activated. The key prefix is specified by the **cscope-keymap-prefix** user option, which is set to **C-c s** by default. See the CScope commands below in the CScope section of this table.
Toggle helm-cscope-mode	**<f11> X C H**	(**pel-toggle-helm-cscope**)	Toggle helm-cscope-mode and its key bindings in current buffer. • This mode is a complement to cscope-mode. • When enabled, the key bindings and commands described in the section below are activated for the current buffer.

• Select xref back-end

The xref system may use several back-end function at the same time. The list of back-end functions is stored in the **xref-backend-functions** variable.
- Of the listed backends, xref will try only those that are appropriate: the backend-ends must verify that they can process the type of file at point.
- It may be useful to register several back-ends when using various types of files, or when more than one backend do different searches on a given file.

The following sections list the helper commands can be used to toggle several of the available xref and non-back-ends.

• Use dumb-jump

Supports a large number of programming and markup languages

dumb-jump allows simple jump to definitions and back for a large number of programming languages (over 40).
- dumb-jump does not use any tag or index database file and does not require preliminary indexing of the source code.
- Although it is simple to use, it only provides a limited set of features: jumping to the definition of a symbol and back. It's easy to use and will do for most small to medium size projects if all you need is being able to jump to symbol definition.
- On Emacs 25.1 and later, dumb-jump acts as a backend for xref using its main two commands bound to **M-.** and **M-,**. It provides other commands that implement equivalent xref functionality but with xref they are obsolete and they do not offer anything not available to what dumb-jump offers when acting as a back-end for xref.

📦Requires the **dumb-jump** external package. PEL activates it when **pel-use-dumb-jump** is set to **t**.
- With PEL you can manually activate dumb-jump xref backend with the **<f11> X B D** key sequence. You can also have dumb-jump automatically activated for buffer of major modes identified in the **pel-modes-activating-dumb-jump** user option.

Toggle use of dumb-jump as xref back-end	**<f11> X B D**	(**pel-xref-toggle-dumb-jump-mode**)	Activate/deactivate dumb-jump mode.

- dumb-jump is a xref back-end that does not use tags files. It uses fast regular expression file search with ag or ripgrep to locate references.
- dumb-jump supports a large (and growing) number of programming languages. For relatively small and medium code base it can perform very well. For very large code base you might get better performance with tags based systems like ggtags, but then you must create the global tag database.
- The dumb-jump mode also activates a set of dumb-jump specific commands on older Emacs versions.

• Use Etags

Supports a large number of programming and markup languages

With the xref-etags-mode, the ref system uses a etags-compliant TAGS database file.
1. First you must create the etags-compliant TAGS file. This can be done is various ways:
   - PEL provides a set of shell script files stored in the **pel/bin directory** that use find or fd and etags or Universal Ctags to create the TAGS file.
   - You can also use ⨕ Projectile to create a TAGS file for the project.
   - See examples toward the end of the table for doing it using commands launched from within Emacs.
   - In most cases you will want to create and store the TAGS file at the root directory of your project (which ⨕ Projectile does) and include the definition taken from all the source files on the directory tree to be listed with relative path.
2. If the Xref-Mode is not already active, turn it on with xref-etags-mode (with PEL you can use **<f11> X B E**)
3. Activate your TAGS file with visit-tags-table (with PEL, you can use <**f11> X t**)

Toggle the Etags xref back-end	**<f11> X B E**	(**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.
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.

☝ This sets the global or buffer local value of tags-file-name:
- Normally M-x visit-tags-table sets the global value of 'tags-file-name'. With a prefix arg, set the buffer-local value instead.
- By default (when **tags-add-tables** user option is set to 'ask') 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.

Description	Keystroke	Function	Note
• **Use GNU Global ggtags** Supports a large number of programming and markup languages	The **GNU Global** is the most powerful tags-based system that supports a lot of programming languages and comes with the the ability to create HTML files for your project that helps you quickly navigate inside source code. • GNU Global integrates with **Universal CTags** and **Pygments** to provide support for several programming languages. • See the **GNU GLOBAL gtags installation instructions in the PEL manual**: it's important to use the instructions to get the full functionality. • To use it you must create the tag files by running gtags at the root of your project. • Once that's done, use **ggtags-mode** in Emacs for one of the source code file. PEL provides the **`<f11> X B G`** key sequence to toggle it. • ggtags-mode does not need to load the (possibly very large) gtags files, which is an advantage compared to the use of other etags/ctags modes. 📦 Requires the **ggtags** external package (and the GNU GLOBAL, Universal CTags and Pygments tools). PEL activates it when the **pel-use-ggtags** user option is set to t. • With PEL, you can also identify major modes that will automatically activate ggtags-mode in the **pel-modes-activating-ggtags** user option list. From within Emacs, you can: 1. Use the ggtags-mode and its special key sequences. 2. Also use the gxref xref back-end which maps the standard ref navigation keys to use the gtags generated database file. • PEL provides the **`<f11> X B g`** key sequence to toggle use of gxref. • 📦 This requires the **gxref** external package PEL activates it when **pel-use-gxref** is **t.**		
Toggle ggtags-mode : GNU Global as xref back-end with extra commands	**`<f11> X B G`**	(**ggtags-mode** &optional ARG)	Toggle Ggtags mode on or off. Uses GNU GLOBAL tags database system. • With a prefix argument ARG, enable Ggtags mode if ARG is positive, and disable it otherwise. • When ggtags-mode is active the ggtags-mode key bindings are activated, providing access to the GNU Global extra functionality. See the ggtags section below.
Toggle the use of GNU Global xref-backend	**`<f11> X B g`**	(**pel-xref-toggle-gxref**)	Toggle activation of the gxref xref-back-end for the current major mode.
• **Use RTags** • Supports C, C++	RTags is a client/server application that indexes C/C++ source code and keeps a persistent database. 📦 Requires the **rtags** external package PEL activates it when **pel-use-rtags** is t.		
Toggle use of RTags	**`<f11> X B R`**	(**pel-xref-toggle-rtags**)	Toggle activation of the rtags xref-back-end for C modes
• **Select xref front-end**	For search that find multiple results, the xref front-end determines how these results are displayed. By default xref display them inside a *xref* buffer., where we can select the line of interest and hit return on it to open the result file/line in a new buffer. It's also possible to display the search results differently, using a different xref front-end.		
	`<f11> X F`	(**pel-xref-select-front-end**)	Prompt user to select one of the following xref front end: • xref, which displays results in a *xref* buffer • helm-xref, which displays the results inside a helm buffer. This provides a large set of helm-related functionality for further searches and actions. • ivy-ref, which displays the results in a simple ivy list
Cross-reference Navigation and look-up commands	Most cross reference and indexing engines share a set of key bindings for the main functions: • **`M-.`** : find and move point to identifier definition • **`M-,`** : move point back to original location before cross-reference find/move • **`M-?`** : find all references to an identifier Some provide more commands.		
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-.`**
		• ⚠️ 🐞The etags-xref-backend does not handle compressed files as it should. Reported in **GNU bug report #44494**. • That problem affects finding references in Emacs library files which are often compressed. • PEL implements a work-around for this bug via the file **pel-etags.el**.
	`M-.`	(**helm-cscope-find-global-definition** SYMBOL)	Find a symbol's global definition using the CScope database. • Show the results in a helm buffer. • Used when **xcscope** and **helm-cscope** are both active.
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.
	`M-,`	(**helm-cscope-pop-mark**)	Pop back to where cscope was last invoked. Used when **helm-cscope** is used.
For ggtags-mode only:			
• **Move to file where navigation starts**	**`M-=`**	(**ggtags-navigation-start-file**)	Move to the file where navigation session starts. ⚠️ While ggtags-mode is active this key binding overrides the binding to **er/expand-region**. ☝️ However, with PEL, you can still access er/expand-region with the **`<f11> . =`** key sequence.
• **Move to next tag marker**	**`C-c M-n`**	(**ggtags-next-mark** &optional ARG)	Move to the next (newer) mark in the tag marker ring (shown in the *Tag Ring* buffer).
• **Move to previous tag marker**	**`C-c M-p`**	(**ggtags-prev-mark**)	Move to previous (older) mark in the tag marker ring (shown in the *Tag Ring* buffer).
For cscope only:			
• **Next symbol**	**`C-c s n`**	(**cscope-history-forward-line-current-result**)	Like (cscope-history-forward-line), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results
• **Next file**	**`C-c s N`**	(**cscope-history-forward-file-current-result**)	Like (cscope-history-forward-file), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results.
• **Previous symbol**	**`C-c s p`**	(**cscope-history-backward-line-current-result**)	Like (cscope-history-backward-line), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results
• **Previous file**	**`C-c s P`**	(**cscope-history-backward-file-current-result**)	Like (cscope-history-backward-file), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results.
• **Move back**	**`C-c s u`**	(**cscope-pop-mark**)	Pop back to where cscope was last invoked.

Description	Keystroke	Function	Note
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.		
FInd references (uses) of 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, some other require a **C-u** prefix argument to request prompt. • To force always prompting set the **xref-prompt-for-identifier** user option to **t**. • Return to original position with **M-,** • ⚠ The default backend for several types of files uses Unix commands find and grep to search over the set of files: a slow operation.
For gtags-mode only:			
• **Find references (uses) of symbol at point**	**M-]**	(**ggtags-find-reference** NAME)	Find all references to the symbol at point using gtags. • With **C-u** prefix, prompt for the symbol. • Available when ggtags-mode is on. • If one found move point to it. • If several are found list them in a *ggtags-global* buffer: select one to jump to it. • Return to original position with **M-,**
• **Find symbol definitions by POSIX regex**	**C-M-.**	(**ggtags-find-tag-regexp** REGEXP DIRECTORY)	List tags matching POSIX REGEXP in DIRECTORY (default to project root). • With a prefix, ask for the directory. • List all found symbol definitions in a *ggtags-global* buffer.
For helm-cscope-mode:			
• **Find all callers of function at point**	**M-@**	(**helm-cscope-find-calling-this-function** SYMBOL)	Display functions calling a function.
• **Find symbol at point in source code**	**M-s**	(**helm-cscope-find-this-symbol** SYMBOL)	Locate a symbol in source code.
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.		

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 operations related to a tags-table created by a previously done by one of xref-backend based search like M-. or M-? that created a list of tags in a *xref* buffer. • 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 work with the following xref backends: elisp, etags, ggtags		
List identifiers defined in a specified source file	**<f11> X l**	(**list-tags** FILE &optional NEXT-MATCH)	Display list of tags that have been detected in a specified source code 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.
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)	Move point to the next definition of currently looked-up symbol (following a tags-based search). • A prefix ARG specifies how many references to move; negative means move back to previous references. • Just **C-u** as a prefix means going back to the first reference found.
Move to previous xref found	• **M-g p** • **M-g M-p**	(**previous-error** &optional N)	Move point to previous reference (from the list of references found by a tags-based search). • Prefix arg N says how many references to move backwards (or forwards, if negative).

Description	Keystroke	Function	Note
Interactively replace identifier in current and next references.	**<f11> X M-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 in 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.
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.
CScope support	CScope is mainly used to index C source code. It also has partial support for C++ and Java. • Although the CScope project is not a=very active in 2020, it can still be used to navigate C source code. • PEL provides commands you can use to quickly activate or deactivate CScope. • When CScope mode is active, a Cscope menu entry is available. Use **<f10>** to open it.		
• **CScope Db control with xcscope**	CScope uses its own database which must be created before you can perform CScope-based searches. • Use the following commands to identify the location of the CScope database and to create it. • See an example describing **how to index the Linux kernel source code tree with CScope** command line.		
Set CScope database directory	**C-c s a**	(**cscope-set-initial-directory** CS-ID)	Set the cscope-initial-directory variable. The cscope-initial-directory variable, when set, specifies the directory where searches for the cscope database directory should begin. This overrides the current directory, which would otherwise be used.
Unset CScope database directory	**C-c s A**	(**cscope-unset-initial-directory**)	Unset the cscope-initial-directory variable.
Create list of files to index	**C-c s L**	(**cscope-create-list-of-files-to-index** TOP-DIRECTORY)	Create a list of files to index. The variable, "cscope-index-recursively", controls whether or not subdirectories are indexed.
Create list and index	**C-c s I**	(**cscope-index-files** TOP-DIRECTORY)	Index files in a directory. • This function creates a list of files to index in cscope.files, and then indexes the listed files stored in cscope.out. • The user option variable, "**cscope-index-recursively**", controls whether or not subdirectories are indexed. It is **t** by default.
Edit list of files to index	**C-c s E**	(**cscope-edit-list-of-files-to-index**)	Search for and edit the list of files to index, the file cscope.files. • If this functions causes a new file to be edited, that means that a cscope.out file was found without a corresponding cscope.files file.
Locate the CScope database directory for the current buffer	**C-c s S**	(**cscope-tell-user-about-directory**)	Display the name of the directory containing the cscope database.
	C-c s T		
	C-c s W		
Open the CScope directory for the current buffer	**C-c s D**	(**cscope-dired-directory**)	Run dired upon the cscope database directory. If possible, the cursor is moved to the name of the cscope database file.
• **CScope commands using xcscope**	📦 These commands require the **xcscope** external package and the **CScope** command line utility. 🛂 PEL activates **xcscope** when **pel-use-xcscope** user option is **t**. You must also install the **CScope** command line utility yourself if it is not present. For these commands, the results are shown inside a *cscope* buffer. This buffer shows the history of CScope operations and results. Each one includes: • Description of the request • CScope database directory used for the operation • Results: filename and each line with a match. • Search time for the operation.		
Find symbol in source	**C-c s s**	(**cscope-find-this-symbol** SYMBOL)	Locate a symbol in source code.
Find symbol global definition (prompts)	**C-c s d**	(**cscope-find-global-definition** SYMBOL)	Find a symbol's global definition.
	C-c s g		
Find symbol definition (without prompting)	**C-c s G**	(**cscope-find-global-definition-no-prompting**)	Find a symbol's global definition without prompting.
Find all assignments to symbol	**C-c s =**	(**cscope-find-assignments-to-this-symbol** SYMBOL)	Locate assignments to a symbol in the source code.
Find all callers of function at point	**C-c s c**	(**cscope-find-functions-calling-this-function** SYMBOL)	Display functions calling a function

Description	Keystroke	Function	Note
Show functions called by function	`C-c s C`	(**cscope-find-called-functions** SYMBOL)	Display functions called by a function.
Locate text string	`C-c s t`	(**cscope-find-this-text-string** SYMBOL)	Locate where a text string occurs.
Run egrep on cscope database	`C-c s e`	(**cscope-find-egrep-pattern** SYMBOL)	Run egrep over the cscope database.
Locate a file	`C-c s f`	(**cscope-find-this-file** SYMBOL)	Locate a file.
Locate all files #including a file	`C-c s i`	(**cscope-find-files-including-file** SYMBOL)	Locate all files #including a file.
• **CScope result buffer**			
Display the *cscope* buffer	`C-c s b`	(**cscope-display-buffer**)	Display the *cscope* buffer.
Toggle automatic display of *cscope* buffer on search results	`C-c s B`	(**cscope-display-buffer-toggle**)	Toggle cscope-display-cscope-buffer, which corresponds to "Auto display *cscope* buffer".
• **CScope navigation**			
Next symbol	`C-c s n`	(**cscope-history-forward-line-current-result**)	Like (cscope-history-forward-line), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results
Next file	`C-c s N`	(**cscope-history-forward-file-current-result**)	Like (cscope-history-forward-file), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results.
Previous symbol	`C-c s p`	(**cscope-history-backward-line-current-result**)	Like (cscope-history-backward-line), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results
Previous file	`C-c s P`	(**cscope-history-backward-file-current-result**)	Like (cscope-history-backward-file), but limited to the current result only. This exists for blind navigation. If the user isn't looking at the *cscope* buffer, they shouldn't be jumping between results.
Move back	`C-c s u`	(**cscope-pop-mark**)	Pop back to where cscope was last invoked.
Extra Cscope commands for xcscope made available when helm-cscope mode is active	The following command and key bindings are available when the helm-cscope mode is active (see above command to control that). • While these commands are available, they mask other commands that use the same bindings. Use the `<f11> X C H` key sequence to turn the mode off and regain access to the previously available command key bindings. 📦 These commands require the **xcscope** and **helm-cscope** external packages and he **CScope** command line utility. 🔄 PEL activates **xcscope** when **pel-use-xcscope** user option is **t**. It enables the command when **pel-use-helm-cscope** is **t**. 🔄 If you want this mode automatically activated in one of the supposed major modes, add the modes to the **pel-modes-activating-helm-cscope** user option.		
Find definition of identifier at point	`M-.`	(**helm-cscope-find-global-definition** SYMBOL)	Find a symbol's global definition using the CScope database. • Show the results in a helm buffer.
Go back to where M-. was last issued	`M-,`	(**helm-cscope-pop-mark**)	Pop back to where cscope was last invoked.
Find all callers of function at point	`M-@`	(**helm-cscope-find-calling-this-function** SYMBOL)	Display functions calling a function.
Find symbol at point in source code	`M-s`	(**helm-cscope-find-this-symbol** SYMBOL)	Locate a symbol in source code.
GNU Global support with ggtags	The **GNU Global** is the most powerful tags-based system that supports a lot of programming languages and comes with the the ability to create HTML files for your project that helps you quickly navigate inside source code. GNU Global integrates with **Universal CTags** and **Pygments** to provide support for several programming languages. • See the **GNU GLOBAL gtags installation instructions in the PEL manual**: it's important to use the instructions to get the full functionality. • To use it you must create the tag files by running gtags at the root of your project. • Once that's done, just **activate ggtags-mode** in Emacs for one of the source code file. PEL provides the `<f11> X B G` key sequence to toggle it. • ggtags-mode does not need to load the (possibly very large) gtags files, which is an advantage compared to the use of other ctags modes. 📦 Requires the **ggtags** external package (and the GNU GLOBAL, Universal CTags and Pygments tools). 🔄 PEL activates it when the **pel-use-ggtags** user option is set to t. You can also identify major modes that will automatically activate ggtags-mode in the **pel-modes-activating-ggtags** user option list.		
Move to definition of symbol at point or to all references of a definition at point ★★★	`M-.`	(**ggtags-find-tag-dwim** NAME &optional WHAT)	Find NAME by context. • If point is at a definition tag, find references, and vice versa. • If point is at a line that matches 'ggtags-include-pattern', find the include file instead. When called interactively with a prefix arg, always find definition tags.
Go back to where M-. was last issued	`M-,`	(**xref-pop-marker-stack**)	Pop back to where M-x xref-find-definitions was last invoked (ie. go back).
Move to file where navigation starts	`M-=`	(**ggtags-navigation-start-file**)	Move to the file where navigation session starts. ⚠️ While ggtags-mode is active this key binding overrides the binding to **er/expand-region**. ☝ However, with PEL, you can still access er/expand-region with the `<f11> . =` key sequence.
Move to next tag marker	`C-c M-n`	(**ggtags-next-mark** &optional ARG)	Move to the next (newer) mark in the tag marker ring (shown in the *Tag Ring* buffer).
Move to previous tag marker	`C-c M-p`	(**ggtags-prev-mark**)	Move to previous (older) mark in the tag marker ring (shown in the *Tag Ring* buffer).
FInd references (uses) of symbol at point	`M-]`	(**ggtags-find-reference** NAME)	Find all references to the symbol at point. • With `C-u` prefix, prompt for the symbol • If one found move point to it. • If several are found list them in a *ggtags-global* buffer: select one to jump to it. • Return to original position with `M-,`
Find symbol definitions by POSIX regex	`C-M-.`	(**ggtags-find-tag-regexp** REGEXP DIRECTORY)	List tags matching POSIX REGEXP in DIRECTORY (default to project root). • With a prefix, ask for the directory. • List all found symbol definitions in a *ggtags-global* buffer.
Save search session in Emacs register See: ∑ **Registers**	`C-c M-SPC`	(**ggtags-save-to-register** R)	Save current search session to register R. • Use `C-x r j` to restore the search session.
Query replace symbols from in gtags	`C-c M-%`	(**ggtags-query-replace** FROM TO &optional DELIMITED)	Query replace FROM with TO on files in the Global buffer. • If not in navigation mode, do a grep on FROM first. ⚠️ The regular expression FROM must be supported by both Global and Emacs.
List history of ggtags symbols searched	`C-c M-/`	(**ggtags-view-search-history**)	Pop to a buffer to view or re-run past searches in a *Ggtags Search History* buffer.
Show symbol definition.	`C-c M-?`	(**ggtags-show-definition** NAME)	Show the definition of specified identifier on the echo line. Acts like eldoc.

Description	Keystroke	Function	Note
Find and list files containing gtags references.	`C-c M-f`	(**ggtags-find-file** PATTERN &optional INVERT-MATCH)	Find files used in the ggtags database whose name match regex PATTERN. Display the names of files found in a *ggtags-global* buffer operating in ggtags-global-mode.
Grep for pattern in gtags results	`C-c M-g`	(**ggtags-grep** PATTERN &optional INVERT-MATCH)	Grep for lines matching PATTERN. • Invert the match when called with a prefix arg **`C-u`**. List all lines not matching. • ⚠ Be careful as this will most probably generate a very large output and that may stall Emacs. If you find yourself in that situation, the easiest way out is to kill the global sub-process. The **htop** application is very useful for that. • Display the names of files found in a *ggtags-global* buffer operating in ggtags-global-mode.
List history of visited locations for the current symbol being searched	`C-c M-h`	(**ggtags-view-tag-history**)	Pop to a *Tags Ring* buffer listing visited locations from newest to oldest. • The buffer is a next error buffer and works with standard commands **`M-n`** 'next-error' and **`M-p`** 'previous-error'.
Query the ID Utils	`C-c M-i`	(**ggtags-idutils-query** PATTERN)	ggtags-idutils-query • This requires the GNU ID Utils.
Open a dired buffer on project root	`C-c M-j`	(**ggtags-visit-project-root** &optional PROJECT)	Visit the root directory of (current) PROJECT in dired. • When called with a prefix **`C-u`**, choose from past projects.
Find reference that is not a definition	`C-c M-o`	(**ggtags-find-other-symbol** NAME)	Find tag NAME that is a reference without a definition.
Browse source code hypertext rendering - create HTML files if required. ★★★	`C-c M-b`	(**ggtags-browse-file-as-hypertext** FILE LINE)	Browse FILE in hypertext (HTML) form. ☝ If a HTML rendering of the code does not exists, prompts to create one and then launch the browser into it. a HTML directory tree is created in the current directory. • The HTML files are created by the htags command line utility that is part of GNU GLOBAL. • ⚠ The creation of the HTML files can take a long time. This is done synchronously, so Emacs is stalled during its execution. The only way to stop it is to kill the htags sub-process. The **htop** application is very useful for that.
Kill buffers visiting files in current project	`C-c M-k`	(**ggtags-kill-file-buffers** &optional INTERACTIVE)	Kill all buffers visiting files in current project.
Delete the Global gtags files	`C-c M-DEL`	(**ggtags-delete-tags**)	Delete file GTAGS, GRTAGS, GPATH, ID etc. generated by gtags.
Creating etags-compliant TAGS files - using pel/bin/etags-xx	PEL provides a set of shell script files stored in the **pel/bin directory**: • `etags-c` : creates a etags compliant TAGS file for C source code files in one of several directory trees. • `etags-cpp` : creates a etags compliant TAGS file for C and C++ source code files in one of several directory trees. • `etags-el` : creates a etags compliant TAGS file for Emacs Lisp source code files in one of several directory trees. • `etags-erl` : creates a etags compliant TAGS file for Erlang source code files in one of several directory trees. • `etags-lisp` : creates a etags compliant TAGS file for Common Lisp source code files in one of several directory trees. • `etags-py` : creates a etags compliant TAGS file for Python source code files in one of several directory trees. Add this directory to your PATH to easily create a TAGS file for the specific programming language. Or use these as examples.		
Creating etags-compliant TAGS files - manually	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. • A simpler way would be to use ∑ **Projectile** which has the ability to create tags file for all source code files inside the project.		
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. ☛ Note: here, shell expands the list of files specified.
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. ☛ Note: here, shell expands the list of files specified.
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. ☝ Don't forget to quote the '**`*.py`**' otherwise your command will expand all file names and you will end up passing a file name as a command to find which will fail.
Create a etags-compliant TAGS file for .py and .pyw Python files in current directory tree	• `M-! rm TAGS` • `M-! find . -type f \( -name "*.py" -or -name "*.pyw" \) -exec etags -a {} \;`		Same as above but using the find -exec option to be able to issue a single command and not use an intermediate file. ⚠ However, you may want to remove the old TAGS file first otherwise new identifiers will be added to the existing TAGS file. • Note the use of the stags -a (--append) option; it is required since etags is executed for each independent file instead of being given the list of all files. • With this method you can execute the same commands where the find first argument identifies another directory tree (instead of '.') . That may be useful to add the identifiers of libraries to the TAGS file of your local project.
	`M-! find . -type f \( -name "*.el" -or -name "*.el.gz" \) -exec etags -a {} \;`		
	`find . -type f \( -name "*.el" -or -name "*.el.gz" \) -print | etags -`		

References — Tags

Topic & Link	Description
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.
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.