Emacs support for Erlang 🚧

Description	Keystroke	Function	Note
Support for the Erlang Programming Language	📦Emacs provides support for Erlang and Erlang Tools via the erlang-mode external package and some other packages. PEL activates Erlang support via the customize user option variable **pel-use-erlang**. It must be set to **t** to activate support for Erlang. 🎛️ Further customization is available via several user options. • PEL customization for Erlang: use **`M-x customize-group pel-pkg-for-erlang`** •		
Editing Erlang Code			
Electric Key in Erlang Source code	The following keys have "*electric*" behaviour and perform special editing tasks to help edit Erlang source code.		
	`,`	(**erlang-electric-comma** &optional ARG)	Insert a comma character and possibly a new indented line. • The variable 'erlang-electric-comma-criteria' states a criterion, when fulfilled a newline is inserted and the next line is indented. • Behaves just like the normal comma when supplied with a numerical arg, point is inside string or comment, or when there are non-whitespace characters following the point on the current line.
	`;`	(**erlang-electric-semicolon** &optional ARG)	Insert a semicolon character and possibly a prototype for the next line. • The variable 'erlang-electric-semicolon-criteria' states a criterion, when fulfilled a newline is inserted, the next line is indented and a prototype for the next line is inserted. Normally the prototype consists of " ->". Should the semicolon end the clause a new clause • header is generated. • The variable 'erlang-electric-semicolon-insert-blank-lines' controls the number of blank lines inserted between the current line and new function header. • Behaves just like the normal semicolon when supplied with a numerical arg, point is inside string or comment, or when there are non-whitespace characters following the point on the current line.
	`>`	(**erlang-electric-gt** &optional ARG)	Insert a greater-than sign, and optionally insert a new line and indent.
Erlang Comments	Erlang uses the % character to identify line comments. It uses the following conventions: • **`%`** - Single percent characters for comments located toward the end of a line of code • **`%%`** - Two percent characters are used for comments starting at indentation level. • **`%%%`** - Three percent characters are used to describe modules and are always placed in the first column		

Comment/un-comment	**`M-;`**	(**comment-dwim** ARG)	Comment line or region with **%** or **%%** style comments depending on the location in the buffer.
			• When no marked region and no comment:
			  • On empty line: insert **%%** comment starter at the proper indentation level.
			  • On line with code: insert **%** comment starter after the code for an end-of-line comment
			• With marked un-commented region:
			  • Comment region (each line is commented)
			• With marked commented region:
			  • removes the comment.
			
			• Call the comment command you want (Do What I Mean).
			  • If the region is active and 'transient-mark-mode' is on, call 'comment-region' (unless it only consists of comments, in which case it calls 'uncomment-region'). Else, if the current line is empty, call 'comment-insert-comment-function' if it is defined, otherwise insert a comment and indent it. Else if a prefix ARG is specified, call 'comment-kill'. Else, call 'comment-indent'.

	`C-c C-c`	(**comment-region** BEG END &optional ARG)	Comment or uncomment each line in the region.
			  • With just **`C-u`** prefix arg, uncomment each line in region BEG .. END.
			  • Numeric prefix ARG means use ARG comment characters.
			  • If ARG is negative, delete that many comment characters instead.
			• The strings used as comment starts are built from '**comment-start**' and '**comment-padding**'; the strings used as comment ends are built from '**comment-end**' and 'comment-padding'.
			• By default, the '**comment-start**' markers are inserted at the current indentation of the region, and comments are terminated on each line (even for syntaxes in which newline does not end the comment and blank lines do not get comments). This can be changed with '**comment-style**'.
			☝If you try this when no region is marked and the **`/* */`** style comments is active, the comment ends on the next space, which is probably not what you want. The command comment-dwim works better.

Indentation	All syntactic indentation control for D is controlled by the CC-Mode logic and provided commands listed below. • Rigid indentation commands are also available and listed at the end of this list. They are also listed in the ∑ Indentation table.		

Indent current line or region
 	
(See also: ∑ Indentation)
	`<tab>`	(**c-indent-line-or-region** &optional ARG REGION)	Indent active region, current line, or block starting on this line.
			
			Behaviour depends on syntactic-indentation mode (enabled by default but can be toggled on/off with the **`<f12> M-i`** key):
			• With syntactic-indentation on (the default):
			  • In Transient Mark mode, when the region is active, reindent the region.
			  • Otherwise, with a prefix argument, rigidly reindent the expression starting on the current line.
			  • Otherwise reindent just the current line.
			  ☝This might seem strange for new Emacs users, but it ends up being very useful. You can type **`<tab>`** anywhere in the line to adjust the indentation of the current line or everything in the marked area if a block is marked.
			• With syntactic-indentation off:
			  • <tab> always indent current line by one level
			  • C-u - <tab> or M- <tab> always un-indent current line by one level
			  • Indenting marked region is done without syntax knowledge and at the same level as previous line.
			☝ If you want to indent rigidly you can use:
			  • (**pel-indent-rigidly &optional N**) (bound to **`C-x <tab>`** and to **`<f11> <tab><tab>`**) to indent the line or region rigidly.
			  • (**tab-to-tab-stop**), bound to **`M-i`** to insert spaces to the next tab stop column.

Indent lines of list after point (See also: ∑ Indentation)	**`C-M-q`**	(**prog-indent-sexp** &optional DEFUN)	Indent the expression after point. When interactively called with prefix, indent the enclosing defun instead.
Indent current function or class	**`C-c C-q`**	(**erlang-indent-function**)	Indent current Erlang function.

Description	Keystroke	Function	Note
Indent a region	**C-M-**	(**indent-region** START END &optional COLUMN)	Indent each nonblank line in the region. • A numeric prefix argument specifies a column: indent each line to that column. • With no prefix argument, the command chooses one of these methods and indents all the lines with it: 1. If 'fill-prefix' is non-nil, insert 'fill-prefix' at the beginning of each line in the region that does not already begin with it. 2. If 'indent-region-function' is non-nil, call that function to indent the region. 3. Indent each line via 'indent-according-to-mode'. ☝When a region is marked you can also use the simple **<tab>** to do the same when syntactic-indentation is active.
Navigation in Erlang code (See also: ⨜ Navigation)	The emac-mode provides commands to navigate across Erlang source code. Most commands are specialization of the normal navigation commands which are described in the table ⨜ Navigation, along with the other commands that are also available. The list below describe the specialized commands only. See the others inside ⨜ Navigation, like the navigation by blocks.		
Go to beginning of statement	**M-a**	(**backward-sentence** &optional ARG)	Go backward to the beginning of an Erlang clause. • With a numerical argument repeat that many times.
Go to the end of statement	**M-e**	(**forward-sentence** &optional ARG)	Go forward to the end of an Erlang clause. • With a numerical argument repeat that many times.
Go to beginning of current function or top-level function	**C-M-a**	(**c-beginning-of-defun** &optional ARG)	Move backward to the beginning of an Erlang function. • Every top level declaration that contains a brace paren block is considered to be a defun. • With a positive argument, move backward that many defuns. A negative argument -N means move forward to the Nth following beginning.
Goto end of current function or top-level function	**C-M-e**	(**c-end-of-defun** &optional ARG)	Move forward to the end of an Erlang function. • With argument, do it that many times. Negative argument -N means move back to Nth preceding end.
Backward to beginning of defun	• **C-M-a** • **C-M-<home>** • **<f6> p**	(**beginning-of-defun** &optional ARG)	Move backward to the beginning of an Erlang clause. • With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun. ☛ Shift marking is available in graphics mode, not in terminal mode (for **C-M-a** and **C-M-<home>**). However **<f6> p** handles Shift-marking fine in terminal mode.
Forward to end of defun	• **C-M-e** • **C-M-<end>**	(**end-of-defun** &optional ARG)	Move forward to end of Erlang function. With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun. ☛ Shift marking is available in graphics mode, not in terminal mode (both keys).
Forward to start of next defun	**<f6> n**	(**pel-beginning-of-next-defun** ARG)	Move forward to the beginning of the next Erlang clause. ☛ Shift marking is available.
Compiling Erlang Code	The following commands are used to compile Erlang source code files to .beam files located in the same directory as the source code. Detected errors are listed in the *erlang* shell opened to compile the files. The buffer shows the location of error and the error description. The following commands are used to navigate to the next or previous detected error.		
Compile code	**C-c C-k**	(**erlang-compile**)	Compile Erlang module in current buffer. • If buffer visiting file was modified and not saved, prompts the user to save it first. • Opens and *erlang* shell, in which the Erlang compile is done with a eshell c() command. • The buffer lists the errors. Hitting <RET> on the error file/line move point to that line in the Erlang file buffer. The <RET> key is bound to (**compile-goto-error** &optional EVENT) • It's also possible to use the next-error and previous error.
Move to next compile error	• **C-x `** • **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 compile error	• **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).
Erlang Shell Command History	The following commands can be used to retrieve previously issued Erlang shell commands at the shell prompt. Note that the shell history is saved inside a file the is restored when opening a new shell: therefore commands from previously opened Erlang shells are also available.		
Next shell command	**M-n**	(**comint-next-input** ARG)	Cycle forwards through Erlang shell input history.
Previous shell command	**M-p**	(**comint-previous-input** ARG)	Cycle backwards through Erlang shell input history, saving input.

Using Man inside Emacs and support Erlang Man pages
(See also: ⨜ Help/Info, ⨜ Shells)
	Emacs provide 2 main commands to display man pages inside buffers.
	• Both of these are much more powerful than the usual man reader available on the shell allowing navigation across man pages and opening hyperlinks.
	• The man command uses the system man utility, while woman is a complete implementation. It has some formatting limitations compared to man but it's very useful in systems where man is not available.
	
	To see Erlang man pages:
	On most systems the Man pages for Erlang are not available to the man utility and therefore not available for man inside Emacs.
	There are several ways this can be remedied:
	• One is to set the MANPATH environment variable to include the directory where these files are located. Then man can be used outside and inside Emacs to access Erlang's man pages. For example the following lines can be stored inside a shell script to do this:
	`MANPATH=`manpath`:/usr/local/Cellar/erlang/22.3.2/lib/erlang/man`
	`export MANPATH`
	• Another way is to customize the Emacs **Man-switches** user option variable to something that includes the same directory. This will add the capability of Emacs man to fin the Erlang's man pages without modifying the capabilities of the parent shell. For example, if we want to use the same directory as the above example we need to set the Man-switches which is normally set to nil to the following value:
	"-M`manpath`:/usr/local/Cellar/erlang/22.3.2/lib/erlang/man"
	
	The second alternative can be used to add other directories for the man pages of other programming languages while leaving the ability to have several shells that have their own value of MANPATH. That might be very useful for someone that uses different versions of Erlang in a system and needs access to the man pages of different versions of Erlang. It becomes possible to run different shells inside Emacs with each having its own value of MANPATH and therefore providing the man pages from different locations. It is also possible to place all of these directories inside the Man-switches or MANPATH and buses man's ability to view several pages for the same topic.		

Open a man page inside an Emacs buffer	• **<f11> ? m** • **⌘-M**	(**man** MAN-ARGS)	Using man pages inside emacs is even better than using it from the shell because: • the links are active and can be followed. When the man page describes a directory or file, emacs will open the file or the directory (in direct mode) when pressing <RET> over the link. • You can navigate easily between sections (n/p will move to the next/previous section) • You can use any of the searches. • You can use any of the options to the man command at the prompt, like the -a option to access all man pages of the same name. Then use **M-n** and **M-p** to move from one to the other page, inside the same buffer. • See all keys available in mode description (Use '**<f1> m**' or '**<f11> ? k m**') to do so.
Open a man page without external man process: woman	**<f11> ? w**	(**woman** &optional TOPIC RE-CACHE)	Open a man page file in Emacs using the woman mode, completely implemented in Emacs Lisp (and therefore without using the external 'man' process). That can be very useful under environments where man is not available (such as basic Windows).
🚧 **TODO**			Create a PEL command to create/update the TAGS file for the current Erlang project

Description	Keystroke	Function	Note
			See “During Search - History previous” in search : it applies to Erlang shell
			Inside the Emacs erlang shell, MFA expansion with theta key does not work the way it works in a pure Erlang shell. Why?

Emacs & Erlang— References

Document	Notes
company-mode ; Modular in-buffer completion framework for Emacs	
The Erlang mode for Emacs	On the erlang.org site. Start here. Describes the 2 files (erlang.el and erlang-start.el) provided by the Erlang mode support, how to set them up for various operating systems. Note, however, that PEL provides the setting for you. It also provides an overview of the various features the package provides. There’s missing information though that I will identify later as I find out how to get the system going. One aspect to learn more is related to the various erlang-electric functions and variables. The variable erlang-electric-commands was set to (erlang-electric-comma erlang-electric-semicolon erlang-electric-gt) at first, which does not include the erlang-electric-newline function. I tried adding erlang-electric-newline and activated it, but that made things worse: the newline was no longer automatic after a -> on a function definition line. Another issue: inside the OS-level erlang shell, we can tab-completion a module:function string, but that does not work inside the emacs erlang shell.
EDTS	EDTS: stands for: The Erlang Development Tool Suite
How to install EDTS	Describes some aspects of EDTS and links that may be useful. Lists the requirements. ⚠️🚧After installing EDTS, I got several compile errors, and had to install the following other modules: - auto-complete (v1.5.1) - have to read doc and configure. And perhaps disable company mode?
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.