Emacs Support for LFE — Lisp Flavoured Erlang 🚧

Description	Keystroke	Function	Note
LFE - Lisp Flavoured Erlang programming Language Support	This table describes Emacs support for **LFE** - Lisp Flavoured Erlang - programming language. 📦 LFE support requires the **lfe-mode** external package. ☑ PEL activates it when the **pel-use-lfe** user-option is turned on (**t**). • PEL provides Lispy support for LFE when the lfe-mode is added to the list specified by **pel-modes-activating-lispy** user-option. See 𝔓𝔏𝔐- **Lispy** 📦 Lispy requires the **lispy** external package. ☑ PEL downloads, installs and activates lispy when the **pel-use-lispy** user option is set to **t**.		

Open this PDF file. See also: ∑ **Help/ Info**	`<f11> SPC C-l <f1>`
	`<f12> <f1>`
		(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the local copy of the 𝔓𝔏 **- LFE** PDF file unless a command prefix (like **`C-u`**) was used. In that case it opens the Github-hosted file instead.

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

∑ **Customize** Emacs LFE support	`<f11> SPC C-l <f3>`
	`<f12> <f3>`
		(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs LFE support: the **lfe** customization group, which controls the settings of the lfe-mode.
			• If OTHER-WINDOW is non-nil (use **`C-u`**), display in another window.

LFE buffer Commands	The lfe-mode behaves like a lisp-mode with the addition of the following commands. We editing a LFE buffer, you can use the lispy-mode which enhance lisp-code editing. See 𝔓𝔏𝔐- **Lispy**.		
Insert brackets pair	• `M-[` • `<f12> [` • `<M-f12> [`	(**lfe-insert-brackets** &optional ARG)	Insert a bracket pair. • With numeric argument, enclose as many S-expressions in brackets. • Leave point after open-bracket. ⚠️ The `M-[` key is is the *only* key binding in the lfe-mode key map. But it is a problematic one when Emacs runs in terminal mode because all function keys starting at F5 are encoded with a sequence that begins with the **`Esc [`** characters. When Emacs is running in terminal mode **`M-[`** is undistinguishable from **`Esc [`**. The end result is that all key prefixes using functions keys starting with F5 no longer work properly! To solve that, PEL does the following: • PEL unbinds the **`M-[`** key when Emacs runs in terminal mode. • PEL adds the **`<f12> [`** key binding in both terminal and graphics mode.

LFE Shell
 	
(**Lisp Flavoured Erlang**)
	`<f12> z`
	`<f11> z C-l`
		(**run-lfe** CMD)	Run an inferior LFE process, input and output via a buffer '*inferior-lfe*'.
			• If 'CMD' is given, use it to start the shell, otherwise:
			'inferior-lfe-program' 'inferior-lfe-program-options' -env TERM vt100.
			👉📦 Requires the lfe-mode package and LFE (Lisp Flavoured Erlang) installed.
			👉☑ PEL activates this when the **pel-use-lfe** user option is set to **t**.

LFE Shell			
	`C-d`	(**comint-delchar-or-maybe-eof** ARG)	Delete ARG characters forward or send an EOF to subprocess. • Sends an EOF only if point is at the end of the buffer and there is no input.
	`RET`	(**comint-send-input** &optional NO-NEWLINE ARTIFICIAL)	Send input to process.
	`<C-down>`	(**comint-next-input** ARG)	Cycle forwards through input history.
	`<C-up>`	(**comint-previous-input** ARG)	Cycle backwards through input history, saving input.
	`<delete>`	(**backward-delete-char-untabify** ARG &optional KILLP)	Delete characters backward, changing tabs into spaces. • The exact behavior depends on 'backward-delete-char-untabify-method'. • Delete ARG chars, and kill (save in kill ring) if KILLP is non-nil. • Interactively, ARG is the prefix arg (default 1) and KILLP is t if a prefix arg was specified.
	`<kp-delete>`		
	`C-c C-a`	(**comint-bol-or-process-mark**)	Move point to beginning of line (after prompt) or to the process mark. • The first time you use this command, it moves to the beginning of the line (but after the prompt, if any). If you repeat it again immediately, it moves point to the process mark. • The process mark separates the process output, along with input already sent, from input that has not yet been sent. Ordinarily, the process mark is at the beginning of the current input line; but if you have used **`C-c SPC`** to send multiple lines at once, the process mark is at the beginning of the accumulated input.
	`C-c C-c`	(**comint-interrupt-subjob**)	Interrupt the current subjob.
	`C-c C-d`	(**comint-send-eof**)	Send an EOF to the current buffer's process.
	`C-c C-e`	(**comint-show-maximum-output**)	Put the end of the buffer at the bottom of the window.
	`C-c C-l`	(**comint-dynamic-list-input-ring**)	Display a list of recent inputs entered into the current buffer.
	`C-c C-m`	(**comint-copy-old-input**)	Insert after prompt old input at point as new input to be edited. • Calls 'comint-get-old-input' to get old input.
	`C-c C-n`	(**comint-next-prompt** N)	Move to end of Nth next prompt in the buffer. • If 'comint-use-prompt-regexp' is nil, then this means the beginning of the Nth next 'input' field, otherwise, it means the Nth occurrence of text matching 'comint-prompt-regexp'.
	`C-c C-o`	(**comint-delete-output**)	Delete all output from interpreter since last input. • Does not delete the prompt.
	`C-c C-p`	(**comint-previous-prompt** N)	Move to end of Nth previous prompt in the buffer. • If 'comint-use-prompt-regexp' is nil, then this means the beginning of the Nth previous 'input' field, otherwise, it means the Nth occurrence of text matching 'comint-prompt-regexp'.
	`C-c C-r`	(**comint-show-output**)	Display start of this batch of interpreter output at top of window. • Sets mark to the value of point when this command is run.
Write interpreter output to specified file	**`C-c C-s`**	(**comint-write-output** FILENAME &optional APPEND MUSTBENEW)	Write output from interpreter since last input to FILENAME. • Any prompt at the end of the output is not written. • If the optional argument APPEND (the prefix argument when interactive) is non-nil, the output is appended to the file instead. • If the optional argument MUSTBENEW is non-nil, check for an existing file with the same name. If MUSTBENEW is 'excl', that means to get an error if the file already exists; never overwrite. If MUSTBENEW is neither nil nor 'excl', that means ask for confirmation before overwriting, but do go ahead and overwrite the file if the user confirms. When interactive, MUSTBENEW is nil when appending, and t otherwise.

Description	Keystroke	Function	Note
	`C-c C-u`	(**comint-kill-input**)	Kill all text from last stuff output by interpreter to point. ☝ Quick way to delete the complete input line after the prompt.
	`C-c C-w`	(**backward-kill-word** ARG)	Kill characters backward until encountering the beginning of a word. • With argument ARG, do this that many times.
	`C-c C-z`	(**comint-stop-subjob**)	Stop the current subjob. ⚠ if there is no current subjob, you can end up suspending the top-level process running in the buffer. If you accidentally do this, use **`M-x comint-continue-subjob`** to resume the process. (This is not a problem with most shells, since they ignore this signal.)
	`C-c C-\`	(**comint-quit-subjob)**	Send quit signal to the current subjob.
	`C-c SPC`	(**comint-accumulate**)	Accumulate a line to send as input along with more lines. • This inserts a newline so that you can enter more text to be sent along with this line. Use RET to send all the accumulated input, at once. The entire accumulated text becomes one item in the input history when you send it.
	`C-M-q`	(**indent-sexp** &optional ENDPOS)	Indent each line of the list starting just after point. • If optional arg ENDPOS is given, indent each line, stopping when ENDPOS is encountered. ☛ Not useful with LFE. Tab works better.
	`<tab>`	(**indent-for-tab-command** &optional ARG)	Indent the current line or region, or insert a tab, as appropriate. • This function either inserts a tab, or indents the current line, or performs symbol completion, depending on 'tab-always-indent'. The function called to actually indent the line or insert a tab is given by the variable 'indent-line-function'. • If a prefix argument is given, after this function indents the current line or inserts a tab, it also rigidly indents the entire balanced expression which starts at the beginning of the current line, to reflect the current line's indentation. • In most major modes, if point was in the current line's indentation, it is moved to the first non-whitespace character after indenting; otherwise it stays at the same position relative to the text. • If 'transient-mark-mode' is turned on and the region is active, this function instead calls 'indent-region'. In this case, any prefix argument is ignored.
	`C-c M-o`	(**inferior-lfe-clear-buffer)**	Delete the output generated by the LFE process. ☛ All lines before the current prompt are deleted from the buffer. The Emacs-maintained history is still available.
	`C-c M-r`	(**comint-previous-matching-input-from-input** N)	Search backwards through input history for match for current input. • (Previous history elements are earlier commands.) • With prefix argument N, search for Nth previous match. • If N is negative, search forwards for the -Nth following match.
	`C-c M-s`	(**comint-next-matching-input-from-input** N)	Search forwards through input history for match for current input. • (Following history elements are more recent commands.) • With prefix argument N, search for Nth following match. • If N is negative, search backwards for the -Nth previous match.
	`C-c .`	(**comint-insert-previous-argument** INDEX)	Insert the INDEXth argument from the previous Comint command-line at point. • Spaces are added at beginning and/or end of the inserted string if necessary to ensure that it's separated from adjacent arguments. • Interactively, if no prefix argument is given, the last argument is inserted. • Repeated interactive invocations will cycle through the same argument from progressively earlier commands (using the value of INDEX specified with the first command). • This command is like 'M-.' in bash.