↖ Topic Index

Emacs support for the Seed7 Programming Language

Description	Keystroke	Function	Note
Seed7 Editing ○ Help & customization • Comments • Template Expansion • Auto-indent ○ Marking • Navigation • Compilation • Seed7 Information	PEL supports for the Seed7 programming language is experimental and not yet documented except for what you see here. 📦 The **seed7-mode** external package is installed when ☑ the **pel-use-seed7** user-option is set to t. • Seed7 files are files with .sd7 and .s7i extensions. The **seed7-mode** supports: • Seed7 code highlighting • Insertion of Seed7 bock or line-end comments. Ability to select which type is inserted by comment-swim. • PEL also provides a command to select the comment style allowing easy selection of different styles of multi-line comments, a feature provided by Emacs that PEL uses and provides an easy selection at prompt. • Seed7 code navigation across function and procedures as well to start/end of blocks inside functions/procedure as well as enum and struct. • imenu support, allowing use of all imenu-based navigation commands and pop-up menus. Identifies callable (functions and procedures), interfaces, enums, structs. • Σ **Speedbar** support and top menu with available commands. (see Σ **Menus**) • Seed7-syntax-aware auto-indentation and auto-fill-mode are supported. • Code keyword expansion to Seed7 statements with ability to jump to next field to fill with **tempo markers** and navigation to those. • outline-minor-mode to list the name of Seed7 callables. See Σ **Outline** for more information. • Invocation of Seed7 compiler tools to perform static analysis or compilation of Seed7 code.		
Last updated on:	2025-06-07		
Open this PDF file. See also: Σ **Help/Info**	`<f11> SPC 7 <f1>` `<f12> <f1>`	(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the **𝔓 - Seed7** local PDF. If the prefix argument (like `C-u` or `M--`) is used, then it opens the remote GitHub hosted raw PDF instead. If the **pel-flip-help-pdf-arg** user-option is set it's the other way around.
Σ **Customize** PEL Seed7 support	`<f11> SPC 7 <f2>` `<f12> <f2>`	(**pel-customize-pel** &optional OTHER-WINDOW)	Customize PEL Seed7 support. • If OTHER-WINDOW is non-nil (use `C-u`), display in another window.
Σ **Customize** Emacs Seed7 support	`<f11> SPC 7 <f3>` `<f12> <f3>`	(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Seed7 support: seed7 • If OTHER-WINDOW is non-nil (use `C-u`), display in another window.
Comments	Several commands are unavailable to insert and manipulate commands, listed in Σ **Comments**. Some are duplicated here for convenience. The ones specific to seed7-mode are listed first.		
Toggle between Seed7 (* block *) and # line style	`C-c ;`	(**seed7-toggle-comment-style** &optional ARG)	Toggle the Seed7 comment style between block and line comments. • Optional numeric ARG, if supplied, switches to block comment style when positive, to line comment style when negative, and just toggles it when zero or left out. Note: the default style for all Seed7 buffers is controlled by the `**seed7-uses-block-comment**' customizable user-option. The default is line style comments.
Insert, realign, comment/uncomment region	`M-;`	(**comment-dwim** ARG)	Insert or realign comment on current line (or region if a region is active). • On a single line, the comment is placed *after* the code. • `C-u M-;` executes comment-kill
Toggle display of comments in buffer or active region See also: Σ **Comments**	`<f11> ; ;`	(**hide/show-comments-toggle** &optional START END)	Toggle hiding/showing of comments in the active region or whole buffer. • If the region is active then toggle in the region. Otherwise, in the whole buffer. 📦 This requires the **hide-comnt.el** package (see Σ **Comments**). ☑ PEL activates it when the **pel-use-hide-comnt** user option is **t**.
Change comment style for buffer	`<f11> ; s`	(**pel-comment-style** &optional CUSTOMIZE)	Select a comment style for the buffer: prompts with the list of available styles, showing the currently used one. Apply the choice to the current buffer. • With `C-u` prefix, open the customize buffer to control selection of the default comment style for all buffers (the **comment-style** user option).

As of Emacs 30, Emacs supports 8 different comment styles, listed here: ➡➡	Emacs supports several comment styles, as specified by the **comment-styles** user-option (which can be modified). Some of these styles only take effect when a region of several lines is comments. By changing the style you can create the boxed comments, for instance and also uncomment the box comment with comment-swim (bound to `M-;`) and then change for another comment style in the same buffer.
	• The style selected by the command only affects the current buffer. It is not persistent. The persistent setting is the **comment-style** user option.
	  • `0 = plain:` Start in column 0 (do not indent), as in Emacs-20
	  • `1 = indent-or-triple:` Start in column 0, but only for single-char starters
	  • `2 = indent:` Full comment per line, ends not aligned
	  • `3 = aligned:` Full comment per line, ends aligned
	  • `4 = box:` Full comment per line, ends aligned, + top and bottom
	  • `5 = extra-line:` One comment for all lines, end on a line by itself
	  • `6 = multi-line:` One comment for all lines, end on last commented line
	  • `7 = box-multi:` One comment for all lines, + top and bottom		

Code Template expansion	The **seed7-mode** supports a set of code keyword expansion to Seed7 statements with ability to jump to next field to fill with **tempo markers** and navigation to these fields to complete the template easily. • Code keyword expansion is performed by the **seed7-complete-statement-or-indent** command, bout to the `<tab>` key. • Type the keyword then type <tab> to expand the keyword into the corresponding code that will be properly indented. There are 2 groups of supported keywords. • The keywords shown in the first part of the table expand to their corresponding code template when the keyword is the only word on the line and point is placed just after the last keyword character.		

Top level or block declarations. Type the keyword at the beginning of the line and hit `<tab>` to expand the corresponding code.				
inc	include statement	**for**	for statement
const	constant declaration	**foru**	for-until statement
var	variable declaration	**fors**	for-step statement
proc	procedure declaration	**fore**	for-each statement
func	function declaration	**foreu**	for-each statement combined with an until condition
funcs	short function declaration	**forek**	for-each-key statement
enum	enum type declaration	**foreku**	for-each-key statement combined with an until condition
struct	struct type declaration	**fork**	for-key statement
case	case statement	**forku**	for-key statement combined with an until condition
if	if statement	**repeat**	repeat - until statement
ife	if statement with an else clause	**while**	while statement
ifei	if statement with an elsif clause		
ifeie	if statement with an elsif and an else clause		

Parameter declarations Also expand with `<tab>`	The second group of keywords are expanded when the keyword precedes a closing parenthesis; they are use to expand the parameter declarations.			
in	Declaration of an in-parameter.	**callbn**	Declaration of a call-by-name parameter.
inout	Declaration of an inout-parameter.	**ref**	Declaration of a reference-parameter.
invar	Declaration of an in-var-parameter.	**val**	Declaration of a value-parameter.

Expand keyword or indent	`<tab>`	(**seed7-complete-statement-or-indent**)	If point follows a valid code keyword properly located, this perform code expansion, leaving point at the first location that must be filled. • In that case you can then type <backtab> to move to the next field that needs to be filled (or has already been filled). Those are **tempo markers** that stay in the buffer until the buffer is closed. If point is located anywhere else indent the line or selected block.
Move to next field	`<backtab>`	(**tempo-forward-mark**)	Move point to the next **tempo marker**, the next template field to fill.
Syntax-aware automatic Indentation	Unless explicitly disabled by setting the **seed7-auto-indent** user-option to nil, the `<tab>` and `<return>` key perform syntax-aware automatic indentation of Seed7 code. The `<return>` key also supports the auto-fill-mode. • The number of columns used for each indentation level is controlled by the **seed7-indent-width** user-option, which defaults to 2. • Emacs can use hard tabs as appropriate when you activate the **indent-tabs-mode**. If it is off Emacs only uses space characters. See the Σ **Indentation** page for more information related to indentation control and commands.		
Auto-fill-mode	The **seed7-mode** supports Emacs auto-fill-mode, useful when typing comments. See the Σ **Fill/Justify** page and the **pel-comment-style** command above.		
Marking	The **seed7-mode** support specialized marking. It is also compatible with other Emancs native and package commands. See Σ **Marking** for more information.		

Description	Keystroke	Function	Note
Mark current callable	`C-M-h`	(**seed7-mark-defun**)	Mark the current Seed7 function or procedure. • Put the mark at the end and point at the beginning. • If point is before or between 2 functions or procedure, mark the next one.
Code Navigation	The **seed7-mode** supports syntax-aware procedure/function as well as block aware navigation commands • PEL provides some extra key bindings to Emacs native navigation commands. • The **seed7-mode** also supports imenu-compliant parsing which enables the ability to use a large set of navigation packages. • See **navigation by symbol definition** in the Σ **Navigation** page for more information. • The **seed7-mode** navigation commands display the name and type of block found when the **seed7-verbose-navigation** user-option is turned on (set to **t**).		

Shift-Selection	If you press and hold the **shift** key while typing a movement command, that sets the mark before moving point (Emacs name for cursor) so that the region extends from the original point to its new position. This is called: **Shift-Selection** .
	• **Shift selection** is supported by some navigation commands, not all. The following symbols are used to identify whether the command supports shifts selection:
	  • ⇩ This command supports shift selection in GUI and terminal mode.
	  • ⇓ This command supports shift selection only in GUI mode.
	  • ⇓ This command supports shift selection in GUI mode and also in terminal mode under some conditions (described in the description cell for the command).
	  • ⇟ This command does **not** support shift selection. Sometimes for this you can first set the mark before moving.
	  • Pressing the Shift key when using the key binding for commands that do not show any of these 3 arrows have no impact on the shift selection (and may be inappropriate for the command).		

Move Point	The following sub-sections describe how to navigate across various types of textual and syntactical entities.		

• **by defun**	The commands move point by Seed7 function and procedure definitions.
	☝ In PEL:
	  • The `<f12>` cursor key mappings use `<up>` and `<down>` to move to the beginning or end of the function, procedure or other blocks.
	  • The `<f6>` cursor key mapping use `<up>` and `<down>` to move to the beginning or end of the function or procedure.
	  • The `<f6>` cursor key mapping use `<right>` and `<left>` to move to the beginning or end of the next/previous function or procedure.
	• The advantage of the `<f6>` and `<f12>` key bindings is they support **Shift-Selection** for Emacs in terminal mode, as opposed to the key bindings that sue the Control key which can only support **Shift-Selection** when Emacs is running in Graphics mode.		

Backward to beginning of defun ⇩	• `<f6> <up>` ⇓ • `C-M-a` • `C-M-<home>` • `C-[ C-a` • `Esc C-a`	(**seed7-beg-of-defun** &optional N SILENT DONT-PUSH-MARK)	Move backward to the beginning of a defun. • With ARG, do it that many times. Negative ARG means move forward to the ARGth following beginning of defun. • Prints the name of the function or procedure in the message area. • On successful move, you can move back to original position by typing `M-`` , `<f6> <f6>` or `<f11> . `` • Supports **Shift-Selection** in graphics mode. `<f6><up>` supports it in terminal mode too.
Forward to end of defun ⇩	• `<f6> <down>` ⇓ • `C-M-e` • `C-M-<end>` • `C-[ C-e` • `Esc C-e`	(**seed7-end-of-defun** &optional N SILENT DONT-PUSH-MARK)	Move forward to next end of defun. • With argument, do it that many times. Negative argument -N means move back to Nth preceding end of defun. • Prints the name of the function or procedure in the message area. • On successful move, you can move back to original position by typing `M-`` , `<f6> <f6>` or `<f11> . `` • Supports **Shift-Selection** in graphics mode. `<f6><down>` supports it in terminal mode too.
Forward to start of next defun ⇩	`<f6> <right>`	(**seed7-beg-of-next-defun** &optional N SILENT DONT-PUSH-MARK)	Move forward to the beginning of the next function or procedure. • With optional argument N, repeat the search that many times. • Move back to previous position with `M-`` , `<f6> <f6>` or `<f11> . `` • Supports **Shift-Selection**.
Backward to end of previous define ⇩ 🚧 will be replaced	`<f6> <left>`	(**pel-end-of-previous-defun** &optional SILENT DONT-PUSH_MARK)	Move backwards to the end of the previous function definition. • Issue user error not find end of previous function unless SILENT is non-nil. • If the end of previous function is found, push the start location to the mark ring unless DONT-PUSH_MARK is non-nil. • Move back to previous position with `M-`` , `<f6> <f6>` or `<f11> . `` • Supports **Shift-Selection**.
Forward to end of current block statement ⇩	`<f12> <down>`	(**seed7-to-block-forward**)	Move forward from the beginning of a Seed7 block to its end. • Supports the Seed7 if/end if, block/end block, case/end case, enum/end enum, for/end for, repeat/until, struct/end struct, while/end while. It also supports moving to the end of a function or a procedure. • Move back to previous position with `M-`` , `<f6> <f6>` or `<f11> . `` • Supports **Shift-Selection**.
Backward to beginning of current block statement ⇩	`<f12> <up>`	(**seed7-to-block-backward**)	Move backward from the end of a Seed7 block to its beginning. • supports the Seed7: if/end if, block/end block, case/end case, enum/end enum, for/end for, repeat/until, struct/end struct, while/end while. It also supports moving to the end of a function or a procedure. • Move back to previous position with `M-`` , `<f6> <f6>` or `<f11> . `` • Supports **Shift-Selection**.
Compilation	The Seed7 source code is either interpreted or compiled. In both cases you can verify it's validity by performing a static check of the code, an operation that does not generate any binary file but perform the same language checking that the compiler will do.		
Static check or compile Seed7 file See Σ **Compilation Mode**	`<f12> c`	(**seed7-compile** &optional *COMPILE*)	Static check current Seed7 file, show errors in compilation-mode buffer. • If optional COMPILE argument set, compile the file to executable instead.

	• For example: type `C-u <f12> c` for compiling the file. Without the `C-u` prefix it just static checks the file, an operation that is much faster.
	• The static analysis is performed by the command identified by the **seed7-checker** user-option, which defaults to s7-check.
	  • You can specify any command with or without its path.
	• The compilation is performed by the command identified by the **seed7-compiler** user-option, which defaults to s7c.
	  • You can specify any command with or without its path.
	• Any detected error is shown in a *compilation* Σ **Compilation Mode** buffer. Use it to navigate to the line of the code in error.		

Emacs & Seed7 — References

Document	Notes
The Seed7 Programming Language	• Seed7 @ Wikipedia • Seed7 Home • Seed7 @ Github • Seed7 Manual • Seed7 Language Reference
	• Seed7 @ reddit • Seed7 @ Rosetta code
Presentations	• The Seed7 Programming Language @ Youtube • The Seed7 Programming Langage Presentation at CPP Vienna @ Youtube • Another speech about the Seed7 Programming Language
	Modern Extensible Languages. Daniel Zingaro, McMaster U. April 11, 2007 (pdf)
Emacs support 🚧 is partial, not yet completed.	• seed7-mode @ Github
Other tools that support Seed7	• **ripgrep** a very fast grep replacement - supports seed7 file types with this pull request accepted April 7 2025 • With this version of ripgrep, you can use **deadgrep** to identify Seed7 files by name in Emacs. See Σ **Grep** • **ugrep** another very fast grep replacement - supports seed7 files with this pull request .