↖ Topic Index

🚧 Emacs support for the Objective-C Programming Language 🚧

Description	Keystroke	Function	Note
Objective-C o Help & Customization • Open alternate-file • Comments o Syntax tools •	Emacs support for **Objective-C** support, objc-mode, is built-in and provided by the cc-mode. 📦 The Emacs built-in **objc-mode** is used and extended with PEL when ☑️ the **pel-use-objc** user-option is set to t. PEL also installs and activates the following external packages when the corresponding user-option is turned on:		
	📦 **flycheck-objc-clang**	☑️ **pel-use-flycheck-objc-clang**	Objective-C support for Flycheck using Clang. When activated PEL automatically activates it when flycheck-mode is activated in an **objc-mode** buffer.
	📦 **objc-font-lock**	☑️ **pel-use-objc-font-lock**	Highlight Objective-C method calls. Note that the default value for the font it defines derive from system faces. You may want to decouple those when selecting a different colour.
	📦 **ccls**	☑️ **pel-use-emacs-ccls-for-objc**	Activate ccls LSP for Objective C.
	Automatic minor mode activation :	☑️ **pel-objc-activates-minor-modes**	If you want to to automatically activate minor modes in **objc-mode** buffers, add the name of the minor mode to the **pel-objc-activates-minor-modes** user option.
Last updated on:	2025-12-04		

Open this PDF file.
See also: ∑ **Help/Info**
	`<f11> SPC C-o <f1>`
	`<f12> <f1>`
		(**pel-help-pdf** &optional OPEN-WEB-PAGE)	Open the 𝔓𝔇 - **Objective-C** 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.

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

∑ **Customize** Emacs Objective-C support	`<f11> SPC C-o <f3>`
	`<f12> <f3>`
		(**pel-customize-library** &optional OTHER-WINDOW)	Customize Emacs Objective-C support: swift
			• If OTHER-WINDOW is non-nil (use **C-u**), display in another window.

Show PEL setup information for the major mode.	`<f11> ? /`
	`<f11> SPC C-o ?`
		(**pel-mode-setup-info** &optional APPEND)	Display Objective-C setup information inside a *pel-mode-info* buffer with buttons providing quick access to the customization buffer of each variable shown. The information shown includes the value and interpretation of control user-options.
			To append information in the buffer instead of clearing the previous content type any prefix argument (such as **C-u**) before the command keystroke.

Opening Files	Specialized file opening commands are listed here. See ∑ **File-mngt** for common file management commands.		

Open file with alternate extension	`M-<f11> M-f M-f`
	`M-<f12> M-f`
		(**pel-open-file-alternate**)	Open a file with same name but an alternate extension: code (.m , .mm) <--> header (.h)
			🎛 The **pel-file-extension-alist** user option defines the extension pairs.
			If no alternate file found, save file basename in the kill ring and prompt for the file name to open.

Comments	Objective-C supports 2 types of comments: **/* Block Comments */** and **// comment to end of line** . Emacs gives electric behaviour to 2 characters:		
/	`/`	(**c-electric-slash** ARG)	Insert a slash character.
	• If the slash is inserted immediately after the comment prefix in a c-style comment, the comment might get closed by removing whitespace and possibly inserting a "*". See the variable 'c-cleanup-list'. • Indent the line as a comment, if: 1. The slash is second of a "//" line oriented comment introducing token and we are on a comment-only-line, or 2. The slash is part of a "*/" token that closes a block oriented comment. • If a numeric ARG is supplied, point is inside a literal, or 'c-syntactic-indentation' is nil or 'c-electric-flag' is nil, indentation is inhibited.		
*****	`*`	(**c-electric-star** ARG)	Insert a star character.
	• If 'c-electric-flag' and 'c-syntactic-indentation' are both non-nil, and the star is the second character of a C style comment starter on a comment-only-line, indent the line as a comment. • If a numeric ARG is supplied, point is inside a literal, or 'c-syntactic-indentation' is nil, this indentation is inhibited. • With this key being electric it becomes easy to type the following two styles of multi-line block comment:		

	`/* Single star`
	` * prefix for`
	` * multi-line`
	` * C comment.`
	` */`
		`/* Two star`
		`** continuation`
		`** prefix for`
		`** multi-line`
		`** C comment.`
		` */`
			• ☝️ When typing the '*' at the beginning of the line, it indents automatically. If another '*' is typed, indentation is set to allow a two-star continuation, otherwise it is placed for a single star continuation.
			• ☝️ When **auto-fill-mode** is active, when you type a comment that would be longer than the line, the line is wrapped and the comment continuation string used is automatically inserted. (toggle it with `<f11> RET`)

Comment/un-comment See also: ∑ **Comments** **With PEL**: Comment the current line with `M-0 M-;`	`M-;`	(**pel-c-comment-dwim** ARG)	Comment line or region with **//** or **/* */** style comments depending on the comment style currently used in the buffer. With numeric argument: comment current line. `M-0 M-;`
	• When no marked region and no comment: • On empty line: insert comment starter at the proper indentation level. Typed again: move it toward end of line. • 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'. • You can configure 'comment-style' to change the way regions are commented: Use `<f12> <f4> M-;` to toggle the comment style.		
	`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.		
Fill current paragraph See also: ∑ **Filling/Justification**	• `M-q` • `<f12> F` • `M-<f12> F` • `<f11> SPC C F`	(**c-fill-paragraph** &optional ARG)	Like `<f11> t f p` but handles **//** and **/* */** style comments. • Optional prefix ARG means justify paragraph as well.

		• If any of the current line is a comment or within a comment, fill the comment or the paragraph of it that point is in, preserving the comment indentation or line-starting decorations (see the '**c-comment-prefix-regexp**' and '**c-block-comment-prefix**' variables for details).
		• If point is inside multiline string literal, fill it. This currently does not respect escaped newlines, except for the special case when it is the very first thing in the string. The intended use for this rule is in situations like the following:
		`char description[] = "\`
		`A very long description of something that you want to fill to make`
		`nicely formatted output.";`
		• If point is in any other situation, i.e. in normal code, do nothing.	

Toggle Comment Style	• `C-c C-k` • `<f12> <f4> M-;`	(**c-toggle-comment-style** &optional ARG)	Toggle the 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.
Toggle display of comments in buffer or active region	`<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. 📦 Requires **hide-comnt.el** ☑️ PEL activates when **pel-use-hide-comnt** user option is **t**.

Description	Keystroke	Function	Note
Getting Syntactic Information	Use the following commands to extract syntactic information from the source code.		
Toggle highlight of method calls	`<f12> M-F`	(**objc-font-lock-mode** &optional ARG)	Minor mode that highlights Objective-C method calls.
Display name of current function	• `C-c C-z` • `<f12> f` • `M-<f12> f`	(**c-display-defun-name** &optional ARG)	Display the name of the current CC mode defun and the position in it. • With a prefix arg, push the name onto the kill ring too.
CC Mode Style Management • **Learn/Modify style used in current buffer**	Automatic indentation, brace format style and several other C stylistic elements are controlled by the CC Mode and the CC mode variables. • You can impose an indentation style by customization. • You can also adjust the style to what is used in the current buffer: Emacs provides the following commands to parse the source code and identify the style it uses. It *learns* the style and sets the style controlling variables from what it detects in the buffer. • ☝Use this to **adapt** to source code written by others and want to continue using the same style, or to **modify** the style. • ☝For the following commands all commands that use a key binding that ends with an upper case letter install the style.		
Show/Modify syntactic context ☝Set style indentation	`C-c C-o`	(**c-set-offset** SYMBOL OFFSET &optional IGNORED)	Change the value of a syntactic element symbol in 'c-offsets-alist'. • SYMBOL is the syntactic element symbol to change and OFFSET is the new offset for that syntactic element. ☝Use this to **modify** a specific style, like how something is indented.
Show syntactic information for current line	`C-c C-s`	(**c-show-syntactic-information** ARG)	Show syntactic information for each syntactic element present *on the current line*. • Display the syntactic information list: style and position highlight the reference position(s) listed as argument to the syntactic list. • Each list starts with a **syntactic symbol** with zero or several reference positions. • With universal argument, inserts the analysis as a comment on that line.
Guess the style used in the current buffer, do not install it	`<f12> <f4> g g`	(**c-guess-buffer-no-install** &optional ACCUMULATE)	Guess the style on the whole current buffer; don't install it. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess the style of the code in the buffer and install it.	`<f12> <f4> g B`	(**c-guess-buffer** &optional ACCUMULATE)	Guess the style on the whole current buffer, and install it. • The style is given a name based on the file's absolute file name. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess style in the region and install it.	`<f12> <f4> g G`	(**c-guess** &optional ACCUMULATE)	Guess the style using the first '**c-guess-region-max**' bytes of the file, and install it. • The **c-guess-region-max** user-option defaults to 50,000 bytes, nil means all buffer. • The style is given a name based on the file's absolute file name. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Guess the style of a region and install it.	`<f12> <f4> g R`	(**c-guess-region** START END &optional ACCUMULATE)	Guess the style on the region and install it. • The style is given a name based on the file's absolute file name. • If given a prefix argument (or if the optional argument ACCUMULATE is non-nil) then the previous guess is extended, otherwise a new guess is made from scratch.
Set buffer style to guessed style and install it.	`<f12> <f4> g I`	(**c-guess-install** &optional STYLE-NAME)	Install the latest guessed style into the current buffer. • This guessed style is a combination of 'c-guess-guessed-basic-offset', 'c-guess-guessed-offsets-alist' and 'c-offsets-alist'. • The style is entered into CC Mode's style system by 'c-add-style'. Its name is either STYLE-NAME, or a name based on the absolute file name of the file if STYLE-NAME is nil.
View Guessed style as a set of Emacs Lisp statements	`<f12> <f4> g ?`	(**c-guess-view** &optional WITH-NAME)	Emit emacs lisp code which defines the last guessed style, so you can put the code into .emacs if you prefer the guessed code. • "STYLE NAME HERE" is used as the name for the style in the emitted code. If WITH-NAME is given, it is used instead. WITH-NAME is expected as a string but if this function called interactively with prefix argument, the value for WITH-NAME is asked to the user.
Toggle preprocessor line indentation	`<f12> <f4> #`	(**c-toggle-cpp-indent-to-body** &optional ARG)	Toggle the C preprocessor indent-to-body feature. When enabled, preprocessor directives which are words in '**c-cpp-indent-to-body-directives**' are indented as if they were statements.
	`C-d`	(**c-electric-delete-forward** ARG)	Delete the following character or whitespace. • If 'c-hungry-delete-key' is non-nil (indicated by "/h" on the mode line) then all following whitespace is consumed. If however a prefix argument is supplied, or 'c-hungry-delete-key' is nil, or point is inside a literal then the function in the variable 'c-delete-function' is called.
	`TAB`	(**c-indent-line-or-region** &optional ARG REGION)	Indent active region, current line, or block starting on this line. • 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.
	`#`	(**c-electric-pound** ARG)	Insert a "#". • If 'c-electric-flag' is set, handle it specially according to the variable 'c-electric-pound-behavior'. If a numeric ARG is supplied, or if point is inside a literal or a macro, nothing special happens.
	`( .. )`	(**c-electric-paren** ARG)	Insert a parenthesis. • If 'c-syntactic-indentation' and 'c-electric-flag' are both non-nil, the line is reindented unless a numeric ARG is supplied, or the parenthesis is inserted inside a literal. • Whitespace between a function name and the parenthesis may get added or removed; see the variable 'c-cleanup-list'. • Also, if 'c-electric-flag' and 'c-auto-newline' are both non-nil, some newline cleanups are done if appropriate; see the variable 'c-cleanup-list'.
	`*`	(**c-electric-star** ARG)	Insert a star character. • If 'c-electric-flag' and 'c-syntactic-indentation' are both non-nil, and the star is the second character of a C style comment starter on a comment-only-line, indent the line as a comment. • If a numeric ARG is supplied, point is inside a literal, or 'c-syntactic-indentation' is nil, this indentation is inhibited.
	• `,` • `;`	(**c-electric-semi&comma** ARG)	Insert a comma or semicolon. • If 'c-electric-flag' is non-nil, point isn't inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: • (a) When the auto-newline feature is turned on (indicated by "/la" on the mode line) a newline might be inserted. See the variable 'c-hanging-semi&comma-criteria' for how newline insertion is determined. • (b) Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. • (c) If auto-newline is turned on, a comma following a brace list or a semicolon following a defun might be cleaned up, depending on the settings of 'c-cleanup-list'.
	`/`	(**c-electric-slash** ARG)	Insert a slash character. • If the slash is inserted immediately after the comment prefix in a c-style comment, the comment might get closed by removing whitespace and possibly inserting a "*". See the variable 'c-cleanup-list'. • Indent the line as a comment, if: 1. The slash is second of a "//" line oriented comment introducing token and we are on a comment-only-line, or 2. The slash is part of a "*/" token that closes a block oriented comment.
	`:`	(**c-electric-colon** ARG)	Insert a colon. • If 'c-electric-flag' is non-nil, the colon is not inside a literal and a numeric ARG hasn't been supplied, the command performs several electric actions: • (a) If the auto-newline feature is turned on (indicated by "/la" on the mode line) newlines are inserted before and after the colon based on the settings in 'c-hanging-colons-alist'. • (b) Any auto-newlines are indented. The original line is also reindented unless 'c-syntactic-indentation' is nil. • (c) If auto-newline is turned on, whitespace between two colons will be "cleaned up" leaving a scope operator, if this action is set in 'c-cleanup-list'.

Description	Keystroke	Function	Note
	• `{` • `}`	(**c-electric-brace** ARG)	Insert a brace. • If ‘c-electric-flag’ is non-nil, the brace is not inside a literal and a numeric ARG hasn’t been supplied, the command performs several electric actions: • (a) If the auto-newline feature is turned on (indicated by "/la" on the mode line) newlines are inserted before and after the brace as directed by the settings in ‘c-hanging-braces-alist’. • (b) Any auto-newlines are indented. The original line is also reindented unless ‘c-syntactic-indentation’ is nil. • (c) If auto-newline is turned on, various newline cleanups based on the settings of ‘c-cleanup-list’ are done.
	`DEL`	(**c-electric-backspace** ARG)	Delete the preceding character or whitespace. • If ‘c-hungry-delete-key’ is non-nil (indicated by "/h" on the mode line) then all preceding whitespace is consumed. • If however a prefix argument is supplied, or ‘c-hungry-delete-key’ is nil, or point is inside a literal then the function in the variable ‘c-backspace-function’ is called.

Emacs & Objective-C — References

Document	Notes
The Objective-C Programming Language	• Objective-C @ Wikipedia • Objective-C Programming @ WikiBooks • Programming with Objective-C @ Apple • Clang Language Extensions • Clang 22 Objective-C Literals
Code Guidelines	• Apple Cocoa Coding Guideline • Google Objective-C Style Guide
Emacs support:	• objc-mode, part of Emacs • Setup Emacs as a modern Objective-C Editor @ medium.com by ensue.ru