Emacs Support for D

Description	Keystroke	Function	Note

Support for the D programming language	Emacs supports the D programming language via an external package: the d-mode. See the reference table below for a list of packages supporting D. PEL attempts to bring all of this together without any required Elisp programming, just setting a set of user options via Emacs customize system.
	
	This table describes PEL support for D programming in Emacs.
	• 📦 PEL uses the d-mode external package which is derived from the Emacs-provided CC Mode.
	• PEL activates D support when the **pel-use-d** user option is set to **t**.
	• Since D is a curly-bracket programming language, it is supported by Emacs CC Mode which brings a set of functionality and controls several editing aspects like indentation, bracket style, electric behaviour of some characters, comment style, hungry delete.
	
	PEL customization for D: Simplifies configuration for editing D source code. This only applies to editing D files, no the files.
	• Emacs customization group: **pel-pkg-for-d**
	  • **pel-use-d**: must be set to **t** to activate PEL support for D.
	  • **pel-d-indentation**: Identifies the number of columns used for indentation. Defaults to 4.
	  • **pel-d-tab-width**: The width of a tab. Defaults to 4. This concept different from indentation: you can have an indentation of 4 and tab width of 8: **`M-i`** will move point to columns that are multiple of 8 **`<tab>`** will indent to a column that is a multiple of 4.
	  • **pel-d-use-tabs**: Whether hard tabs are used in indentation or not: **t**: tabs are used, **nil**: only spaces are used. Default: **nil**.
	  • **pel-d-backet-style**: The bracket/indentation style supported by the electric keys. One of the values supported by Emacs (also possible to define your own with Elisp code). Default to “bsd”.
	• Emacs customization group: **pel-pkg-for-cc**. Applies to all CC Mode related modes (like d-mode).
	  • **pel-cc-auto-newline**: Whether automatic newline mode is active on all CC Mode (including d-mode).
	• You can change each of the PEL default user option by using **`M-x customize`** then browse or search the specific user option or use the **`M-x customize-option`** command and specifying the user option on the prompt.
	
	With PEL, a new buffer will be configured with the settings identified by the PEL customization for D listed above. You can modify them for each buffer using the commands shown below and view their settings for the current buffer with the **`<f12> M-?`** command. None of the commands below change PEL default; they change the value for the current buffer only.		

Display current Mode settings	• **`<f12> M-?`**
	• **`<f11> SPC D M-?`**
		(**pel-cc-mode-info**)	Display information about current CC mode derivative for the current buffer.
			• Example of the information displayed (which reflects PEL’s defaults):
			`-UUU:----F1  hello.d        Top (43,0)     (D//la WK Abbrev) ----------`
			`d-mode state:`
			`- Indent width     : 4`
			`- Tab width        : 4`
			`- Indenting with   : spaces only`
			`- Bracket style    : bsd`
			`- Comment style    : Line comments: //`
			`- Electric chars   : active: #*/(){}:;,`
			`- Auto newline     : on`
			`- Syntactic indent : on`
			`- Hungry delete    : off, but the F11-⌦ and F11-⌫ keys are available.`

Toggle Electric state	• **`C-c C-l`** • **`<f12> M-e`** • **`<f11> SPC D M-e`**	(**c-toggle-electric-state** &optional ARG)	Toggle the electric indentation feature done with the electric character keys. • Optional numeric ARG, if supplied, turns on electric indentation when positive, turns it off when negative, and just toggles it when zero or left out.
Toggle auto-newline insertion mode	• **`C-c C-a`** • **`<f12> M-RET`** • **`<f11> SPC D M-RET`**	(**c-toggle-auto-newline** &optional ARG)	Toggle **auto-newline** feature. • Optional numeric ARG, if supplied, turns on auto-newline when positive, turns it off when negative, and just toggles it when zero or left out. • Turning on auto-newline automatically enables ***electric indentation***. • When the auto-newline feature is enabled (indicated by "/la" on the mode line after the mode name) newlines are automatically inserted after special characters such as brace, comma, semi-colon, and colon.
Set indentation style	• **`C-c .`** • **`<f12> M-s`** • **`<f11> SPC D M-s`**	(**c-set-style** STYLENAME &optional DONT-OVERRIDE)	Set the bracket/indentation style for the current buffer. • Prompts for the name. • Supports tab completion (so use tab to see the list). Can be one of the values supported by Emacs but you can also add your customized mode with some Emacs Lisp code.
Toggle syntactic indentation	• **`<f12> M-i`** • **`<f11> SPC D M-i`**	(**c-toggle-syntactic-indentation** &optional ARG)	Toggle syntactic indentation. • Optional numeric ARG, if supplied, turns on syntactic indentation when positive, turns it off when negative, and just toggles it when zero or left out. • When syntactic indentation is turned on (the default), the indentation functions and the electric keys indent according to the syntactic context keys, when applicable. • When it’s turned off, the electric keys don’t reindent, the indentation functions indents every new line to the same level as the previous nonempty line, and M-x c-indent-command adjusts the indentation in steps specified by ‘c-basic-offset’. The indentation style has no effect in this mode, nor any of the indentation associated variables, e.g. ‘c-special-indent-hook’.
Toggle Comment Style	• **`C-c C-k`** • **`<f12> M-;`** • **`<f11> SPC D 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. • Only the **`//`** and **`/* */`** styles are supported. The **`/+ +/`** comments is not supported.
Toggle Hungry Delete mode	• **`<f12> M-DEL`** • **`<f11> SPC D M-DEL`**	(**c-toggle-hungry-state** &optional ARG)	Toggle hungry-delete-key feature. Affect **`<DEL>`** and **`C-d`** keys. • Optional numeric ARG, if supplied, turns on hungry-delete when positive, turns it off when negative, and just toggles it when zero or left out. • When the hungry-delete-key feature is enabled (indicated by "/h" on the mode line after the mode name) the delete key gobbles all preceding whitespace in one fell swoop. ☝More commands can be used to perform hungry delete. See the section on hungry delete below.
Open Line in Context (See also: ∑ Whitespace)	• **`<f12> RET`** • **`<f11> SPC D RET`**	(**c-context-open-line**)	Insert a line break suitable to the context and leave point before it. • This is the ‘**c-context-line-break**’ equivalent to ‘**open-line**’, which is normally bound to **`C-o`**. See ‘c-context-line-break’ for the details.
Electric Keys and Keywords	The following characters have special meaning when the electrical state is active in a buffer using d-mode.		
	#	(**c-electric-pound** ARG)	Insert a "#". • If ‘c-electric-flag’ is set, handle it specially according to the variable ‘c-electric-pound-behavior’, which can only be nil or ‘alignleft’. If a numeric ARG is supplied, or if point is inside a literal or a macro, nothing special happens. • D does not use the pound character much. It only uses it for #line statements.
	• **`(`** • **`)`**	(**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’.

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.
	:	(**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'.
	• **;** • **,**	(**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'.
D comments	2 more characters have electric behaviour: / and * to help support comments in D. D supports 3 types of comments: • Block Comments: **/*** comment ***/** • Line Comments: **//** comment to end of line • Nesting Block Comments: **/+** nesting /+comments +/ can span multiple lines and surround // style comment **+/** • Documentation Comments: Use several prefixes: **///** , **/**** , and **/++**		
	/	(**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 it becomes easy to type the following two styles of multi-line block comment:
			`/* Two star`
			`** continuation`
			`** prefix for`
			`** multi-line`
			`** C comment.`
			`*/`
			`/* Single star`
			`* 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.

Comment/un-comment	**M-;**	(**comment-dwim** ARG)	Comment line or region with **//** or **/* */** style comments depending on the comment style currently used in the buffer. • 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: see **<F12> M-;** to toggle the comment style.

Fill current paragraph
(See also: ∑ Filling/ Justification)
	• **M-q**
	• **<f12> f**
	• **<f11> SPC D f**
		(**c-fill-paragraph** &optional ARG)	Like **<f11> t f p** but handles **//** and **/* */** style comments.
			• 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.
			• Optional prefix ARG means justify paragraph as well.

Toggle subword-mode (See also: ∑ Text Modes)	• **<f11> t m b** • **<f12> M-b** • **<f11> SPC D M-b**	(**subword-mode** &optional ARG)	Toggle subword-mode: a minor mode that treats sections of camelCase and PascalCase as distinct words. • With a prefix argument ARG, enable Subword mode if ARG is positive, and disable it otherwise. ☝Since D naming convention promotes the use of camelCase for functions, enums, constants and variables and PascalCase for types, using the subword-mode allows you to move into, delete, transpose the section of the words with the corresponding word commands.

Description	Keystroke	Function	Note

Hungry Deletion of Whitespace	The CC mode provides two commands that can perform “hungry whitespace deletion” that can also be used in every mode.
	• ☝ PEL provides the convenient keys with the **<f11>** prefix keys for those 2 commands, available in **all** modes.
	• In modes compatible with the CC Mode (e.g. for C, C++, D, Java, Pike, etc..) it is also possible to activate the Hungry Delete Mode to modify the behaviour of the simple **<DEL>** and **C-d**, to perform hungry deletions. That’s not currently supported in other modes.
	  • When the Hungry Delete Mode is on, the mode-line displays a ‘h’ to the right of the ‘//l’ indication of electric mode.
	  • The Hungry Mode also activates the key prefixes below that start with **C-c**. They are listed but remember they are only available once the Hungry state mode is activated (and that can only be done in modes that are CC Mode compatible).
	  • In modes derived from CC Mode you can also activate the hungry state to make standard delete commands delete hungrily, but that does not work for other modes. PEL provides the **<f12> M-DEL** key for those modes, like the D Mode. See above.		

Delete preceding char or all preceding whitespace. (See also: ∑ Cut & Paste)	• **C-c DEL** • **C-c ⌫** • **C-c C-⌫** • **C-c <C-backspace>** • **C-c C-DEL** • **<f11> ⌫**	(**c-hungry-delete-backwards**)	Delete the preceding character or all preceding whitespace back to the previous non-whitespace character. ⌨☛ In terminal mode, even though **C-⌫**, **<C-backspace>** and **C-DEL** are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized. ☝With PEL, the **<f11> ⌫** binding is always available, in all modes. The other keys are only available in modes derived from the CC Mode. This prevents conflicts with other modes that may use the popular C-c bindings.
Delete next char or all following whitespace. (See also: ∑ Cut & Paste)	• **C-c C-d** • **C-c ⌦** • **C-c C-⌦** • **C-c <C-delete>** • **<f11> ⌦**	(**c-hungry-delete-forward**)	Delete the following character or all following whitespace up to the next non-whitespace character. ⌨☛ In terminal mode, even though **C-⌦** and **<C-delete>** are not available, they are mapped to the non-control key so attempting to type them end up invoking the command anyway because the first key bindings are recognized. ☝With PEL, the **<f11> ⌦** binding is always available, in all modes. The other keys are only available in modes derived from the CC Mode. This prevents conflicts with other modes that may use the popular C-c bindings.
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**	(**indent-pp-sexp** &optional ARG)	Indent each line of the list starting just after point, or pretty-print it. • A prefix argument (**C-u**) specifies pretty-printing. Pretty-printing essentially uses more lines as it places the beginning of each list on a new line.
Indent current function or class	**C-c C-q**	(**c-indent-defun**)	Indent the content of the current top-level function or class. Leaves point unchanged.

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.

Non Syntactic Indentation	Emacs provides the following command to indent without regards to semantics. More information on indentation is available in the ∑ Indentation table.		
Insert spaces or tabs to next defined tab-stop column (See also: ∑ Indentation)	**M-i**	(**tab-to-tab-stop**)	Insert spaces or tabs to next defined tab-stop column. • The exact location of the next tab stop is identified by the value of the **tab-stop-list** and **tab-width** for the current buffer.
Indent/Unindent rigidly (See also: ∑ Indentation)	• **C-x <tab>** • **<f11> <tab> <tab>**	(**pel-indent-rigidly &optional N**) — — — — — — — — — — — — — ✂ PEL uses the above instead of the standard: (**indent-rigidly** START END ARG &optional INTERACTIVE)	Indent rigidly the marked region or current line N times. • **If a region is marked**, it uses ‘indent-rigidly’ and provides the same prompts to control indentation changes. • **If no region is marked**, it operates on current line(s) identified by the numeric argument N (or if not specified N=1): • N = [-1, 0, 1] : operate on current line • N > 1 : operate on the current line and N-1 lines below. • N < -1 : operate on the current line and (abs N) -1 lines above. ✂ PEL rebinds this key, but it extends the functionality: pel-indent-rigidly uses indent-rigidly, described below the dashed line. — Indent all lines starting in the region. • If called interactively with no prefix argument, activate a transient mode in which the indentation can be adjusted interactively by typing **<left>**, **<right>**, **<S-left>**, or **<S-right>**. — These commands activate a transient mode where Emacs prompts for extra keys to control how to indent. Indenting and un-indenting is possible. The capabilities are controlled by the variable *indent-rigidly-map* with by default provides: • **S-<right>** indent-rigidly-right-to-tab-stop • **S-<left>** indent-rigidly-left-to-tab-stop • **<right>** indent-rigidly-right • **<left>** indent-rigidly-left Typing any other key deactivates the transient mode. ⚠ Since cua-mode uses **C-x**, to invoke this command when cua-mode is active, type it really fast or type **C-x C-x <tab>** (or use the PEL binding **<f11> <tab> <tab>**).

Description	Keystroke	Function	Note
Insert an indented line below current line (See also: ∑ Indentation)	• `M-<RET>` • `<f11> <tab> <RET>`	(**pel-newline-and-indent-below**)	Insert an indented line just below current line. • This does the same as the normal **`<RET>`** in d-mode.
Marking	Emacs provides the following command to quickly mark the whole content of the current function. More mark commands exists, see the ∑ Marking table.		
Mark the complete function body (See also: ∑ Marking)	`C-M-h`	(**c-mark-function**)	Mark complete function. • Put mark at end of the current top-level declaration or macro, point at beginning. • If point is not inside any then the closest following one is chosen. Each successive call of this command extends the marked region by one function. • A mark is left where the command started, unless the region is already active (in Transient Mark mode). • As opposed to C-M-a and C-M-e, this function does not require the declaration to contain a brace block.

Emacs & D— References

Document	Notes
The D Programming Language	
D (programming language) - Wikipedia	Overview of D
D Home Page	
D Home Page - Documentation	Links to the Language Reference, Library Reference, Command-line Reference, Feature Overview and Articles.
DUB - The D Package Registry	Browsable/searchable list of packages

The D Style - D Code Guideline	This document provides a set of style conventions promoted by the D community. Several items in this guideline identify stylistic aspects that can be configured in Emacs. Some of them are listed here:
	• **Indentation**:
	  • spaces instead of tabs ➢
	  • indentation level: 4 columns ➢ c-basic-offset = 4
	• **Line Length** : soft limit of 80, hard limit of 120. They can exceed 80 columns but never 120.
	• **Brackets style**:
	  • Use the Allman style (also called BSD style) where each brace is on their own line. ➢ add: (d-mode . "bsd") to c-default-style
	• **Whitespace in statements**:
	  • 1 space after **for**, **foreach**, **if**, **while** and **version** keyword and the opening parenthesis: `if (x) { ... }`
	  • 1 space between binary operators, assignments, casts, lambdas.
	  • No space between unary operators, after assert, function calls, function definition name.
	• **Naming Conventions**:
	  • Constant, enums, variable and function names should be camelCased.
	  • User defined type names should be PascalCased.

The Next Big Programming Language You've Never Heard Of | WIRED - 2014	D is a very nice language, unfortunately it never got the attention could have got if it had some big corporate backup. Interview with Andrei Alexandrescu discussing his encounter with Walter Bright and the D language.

Emacs Support for D	Support for D for Emacs is based on:
	• Emacs D Mode
	• Code completion support that uses:
	  • A completion front end, either:
	  • Auto-Complete based using ac-dcd.
	  • Company based using company-dcd.
	  • Both of these depend on flycheck-dmd-dub, which uses DCD, the D Completion Daemon, written in D.
	  • Both require/use flycheck
	• D Unit test support: flycheck-d-unittest

Emacs D Mode	The main support for D. Available on MELPA as d-mode. The d-mode is based on cc-mode.
ac-dcd : Auto Complete D Code Completion via DCD backend	Available on MELPA as ac-dcd. • This project also recommend using yasnippet and popwin. (require 'ac-dcd) (add-to-list 'ac-modes 'd-mode) (add-hook 'd-mode-hook #'ac-dcd-setup)
Company-DCD - Company D Code Completion via DCD backend	Available on MELPA as company-dcd. • DCD is the D Completion Daemon (DCD @ Github).
flycheck-dmd-dub	Available from melba as flycheck-dmd-dub. • Flycheck support for D: reads D library dependency information from DUB (the D Package Registry). • To use it you must install DCD separately (see instructions on the DCD Github page). • See also the DUB DCD page which has the same info as GitHub but also has internal documentation of the D code interfaces down to the source code. • On macOS, the dcd-client and dcd-server commands can be installed with Homebrew.
D Unit Test support: flycheck-d-unittest	Available on MELPA as flycheck-d-unittest. • Runs D unit test with "dmd -uninttest and -main options". • Takes advantage that D has built-in syntax and dmd support for unit test builds and runs. • The project has a wiki page, "Start D with Emacs", describing how to install Emacs support for D (but only describes d-mode and flycheck-d-unittest)
yasnippets for D	I have found the following: • Per Nordlöw snippets for D
Emacs CC Mode	The d-mode is based on the CC Mode. The CC Mode, a collection of libraries, provides support for several curly-bracket programming languages like C, C++, Java, Objective-C, Pike, AWK and it also applies to D. Several features of the CC Mode are used for the D support, so it's useful to be aware of them.
GNU Emacs CC Mode Manual	D is a curly-bracket programming language and therefore supported by Emacs CC Mode. It controls: • whether hard tabs or spaces are used: indent-tabs-mode • the number of columns per tab: tab-width • the indentation style (see indentation style meanings): c-default-style a-list with an entry for D PEL provides user options to activate the use of D in Emacs (pel-use-D) and user options for the tab, style to use and what CC modes are activated by default.