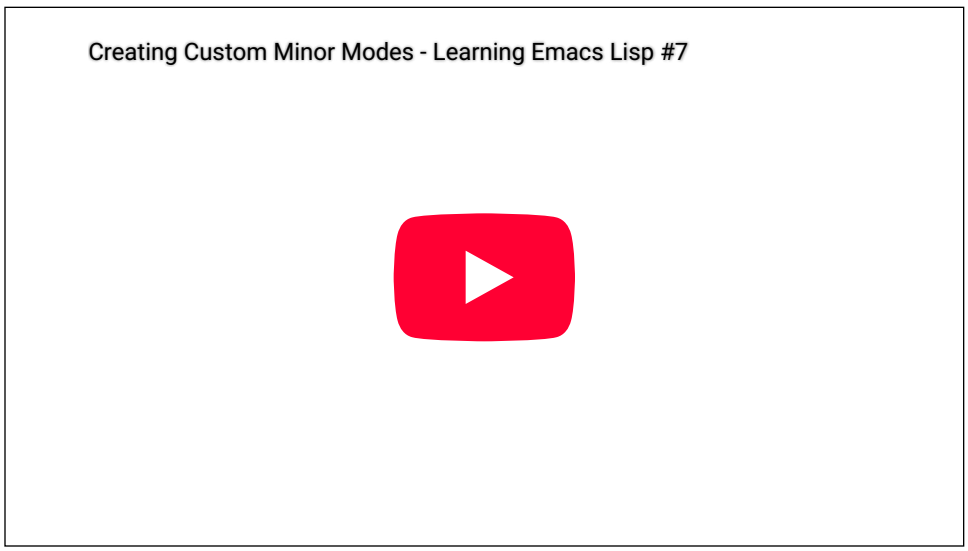
Creating a Custom Minor Mode



¶ What will we learn?

Today we're going to learn about Emacs' concept of modes and then walk through some practical examples of creating a custom minor mode.

We'll be continuing with the `dotcrafter.el` project that we started working on earlier in this series! At this point we have full functionality for managing our dotfiles configuration both from within Org Mode files and with individual files that must be symbolically linked back into your home directory.

The minor mode that we'll write today will enable us to automatically "tangle" our Org-based configuration files each time we edit one of them and then link any new output configuration files into our home directory. We'll also be able to set up a custom key map for dotcrafter commands which will only be activated when the mode is turned on!

If you find this guide helpful, please consider supporting System Crafters via the links on the How to Help page!

¶ What is a "mode"?

Much of the user-facing functionality in Emacs is provided by **major** and **minor** modes.

- A **major** mode is the main mode for a buffer, there can only be one active at any time (example: `org-mode`)
- A **minor** mode provides supporting functionality either in a specific buffer or globally in Emacs (example: `auto-fill-mode`)

Both major and minor modes can enable certain types of functionality when they are activated:

- A key map that is only active when the mode is enabled
- A mode line indicator showing the mode is active
- A hook that is executed so that other functions or modes can be activated along with the mode
- Arbitrary code that gets executed when the mode is activated or deactivated

Any mode that is defined will have a command that enables or disables it which can be invoked interactively or in Emacs Lisp:

```
M-x org-mode

(auto-revert-mode 1)
```

¶ Writing a basic minor mode

A minor mode is basically a bit of Emacs Lisp code that conforms to a set of conventions on how minor modes should be defined. Here are the high level things that a custom minor mode needs to work correctly in Emacs:

- A variable for the mode that ends in `-mode` like `dotcrafter-mode`. This variable will be set to `nil` when the mode is not active or any other value to indicate that it is active. If the minor mode is meant to be buffer-local, the `make-variable-buffer-local` function can be used to make the mode variable buffer-local.
- A command for enabling and disabling the mode which has the same name as the mode variable (e.g. `dotcrafter-mode`). It will accept one optional argument which indicates whether the mode should be enabled or disabled, typically `t` or a positive number to activate the mode or a non-positive number (0 or less than 0) to deactivate the mode.

  When the optional parameter is omitted, assume this means that the mode should be activated when the function is called in Emacs Lisp or that the mode should be toggled when invoked interactively by the user.

- If desired, a keymap for the mode can be enabled by adding an entry to the `minor-mode-map-alist` which registers. The entry will be a pair containing the minor mode's variable symbol and the keymap to be used for that mode.
- If desired, a "lighter" for the minor mode can be displayed in the mode line by adding an entry to the `minor-mode-alist`. The entry will be a pair containing the minor mode's variable symbol and the string to display in the mode line.

The command that is used for enabling and disabling the minor mode can then make whatever changes are needed to the buffer-local or global state of Emacs to enable the mode's behavior.

Note that the mode command will have to check the mode variable to see if it is already enabled before enabling/disabling the mode functionality!

¶ Example: Creating a simple minor mode from scratch

Let's follow the conventions we talked about and write a simple minor mode from scratch! Our minor mode, `dotcrafter-basic-mode`, will simply set a keymap that can be used when this mode is active and also update the mode line with the string "dotcrafter" to indicate its state.

```
(make-variable-buffer-local
  (defvar dotcrafter-basic-mode nil
    "Toggle dotcrafter-basic-mode."))

(defvar dotcrafter-basic-mode-map (make-sparse-keymap)
  "The keymap for dotcrafter-basic-mode")

;; Define a key in the keymap
```

```
(define-key dotcrafter-basic-mode-map (kbd "C-c C-. t")
  (lambda ()
    (interactive)
    (message "dotcrafter key binding used!")))

(add-to-list 'minor-mode-alist '(dotcrafter-basic-mode " dotcrafter"))
(add-to-list 'minor-mode-map-alist (cons 'dotcrafter-basic-mode dotcrafter-basic-mode-map))

(defun dotcrafter-basic-mode (&optional ARG)
  (interactive (list 'toggle))
  (setq dotcrafter-basic-mode
        (if (eq ARG 'toggle)
            (not dotcrafter-basic-mode)
          (> ARG 0)))

  ;; Take some action when enabled or disabled
  (if dotcrafter-basic-mode
      (message "dotcrafter-basic-mode activated!")
    (message "dotcrafter-basic-mode deactivated!")))
```

Note that this doesn't perfectly follow all of the conventions: minor mode commands should also accept numeric prefix arguments for controlling whether the mode gets enabled or disabled, but it's not a hard requirement.

Emacs Lisp Manual: Conventions for Writing Minor Modes

¶ Creating a Hook

One useful thing you might want to provide with your minor mode is a hook. A hook is a variable that stores a list of functions to be invoked when *something* happens, like when a mode is activated.

Defining a hook is easy, you just define a variable that is set to `nil` or an empty list:

```
(defvar dotcrafter-basic-mode-hook nil
  "The hook for dotcrafter-basic-mode.")
```

To execute a hook, use the `run-hooks` function:

```
(run-hooks 'dotcrafter-basic-mode-hook)
```

Right now it doesn't do anything! We need to add a function to the hook with `add-hook`:

```
(add-hook 'dotcrafter-basic-mode-hook (lambda () (message "Hook was executed!")))
```

Now we can update our basic minor mode to invoke this hook:

```
(defun dotcrafter-basic-mode (&optional ARG)
  (interactive (list 'toggle))
  (setq dotcrafter-basic-mode
        (if (eq ARG 'toggle)
            (not dotcrafter-basic-mode)
          (> ARG 0)))

  ;; Take some action when enabled or disabled
  (if dotcrafter-basic-mode
        (message "dotcrafter-basic-mode activated!")
    (message "dotcrafter-basic-mode deactivated!"))

  ;; Run any registered hooks
  (run-hooks 'dotcrafter-basic-mode-hook))
```

You can define as many hooks as you need for your minor mode! It doesn't have to just be the hook for when the mode gets activated.

NOTE: Major modes should use the `run-mode-hooks` function instead! There is extra logic to be used in that case. `run-hooks` is fine for minor mode hooks.

¶ Using the `define-minor-mode` macro

Learning how to create your own minor mode from scratch is useful to show how simple they really are in practice, but once you understand it, you shouldn't have to do it that way every time.

It is much easier to use the `define-minor-mode` macro to define a minor mode since it does most of the work for you!

Here's an example of defining a mode called `dotcrafter-mode` that is exactly like `dotcrafter-basic-mode` except for how it is defined:

```
(define-minor-mode dotcrafter-mode
  "Toggles global dotcrafter-mode."
  nil   ; Initial value, nil for disabled
  :global t
  :group 'dotfiles
  :lighter " dotcrafter"
  :keymap
  (list (cons (kbd "C-c C-. t") (lambda ()
                                  (interactive)
                                  (message "dotcrafter key binding used!"))))

  (if dotcrafter-mode
      (message "dotcrafter-basic-mode activated!")
    (message "dotcrafter-basic-mode deactivated!")))

(add-hook 'dotcrafter-mode-hook (lambda () (message "Hook was executed!")))
(add-hook 'dotcrafter-mode-on-hook (lambda () (message "dotcrafter turned on!")))
(add-hook 'dotcrafter-mode-off-hook (lambda () (message "dotcrafter turned off!")))
```

What we get:

- `dotcrafter-mode` command
- `dotcrafter-mode` variable
- `dotcrafter-mode-hook` (also `dotcrafter-mode-on-hook` and `dotcrafter-mode-off-hook`)
- `dotcrafter-mode-map`
- Lighter and mode map registration

Let's take a look at the generated `dotcrafter-mode` command using `C-h f`!

Emacs Lisp Manual: Defining Minor Modes

¶ Example: Writing real behavior for `dotcrafter-mode`

Now let's take what we've learned and set up `dotcrafter-mode` so that we will automatically tangle and update configuration target files for any Org Mode file that lives inside of our dotfiles folder.

```
(defcustom dotcrafter-keymap-prefix "C-c C-."
  "The prefix for dotcrafter-mode key bindings."
  :type 'string
  :group 'dotfiles)

(defcustom dotcrafter-tangle-on-save t
  "When t, automatically tangle Org files on save."
  :type 'boolean
  :group 'dotfiles)

(defun dotcrafter-tangle-org-file (&optional org-file)
  "Tangles a single .org file relative to the path in
dotfiles-folder.  If no file is specified, tangle the current
file if it is an org-mode buffer inside of dotfiles-folder."
  (interactive)
  ;; Suppress prompts and messages
  (let ((org-confirm-babel-evaluate nil)
        (message-log-max nil)
        (inhibit-message t))
    (org-babel-tangle-file (expand-file-name org-file dotcrafter-dotfiles-folder))
    ;; TODO: Only update files that are generated by this file!
    (dotcrafter-link-config-files)))

(defun dotcrafter--org-mode-hook ()
  (add-hook 'after-save-hook #'dotcrafter--after-save-handler nil t))

(defun dotcrafter--after-save-handler ()
  (when (and dotcrafter-mode
             dotcrafter-tangle-on-save
             (member (file-name-nondirectory buffer-file-name) dotcrafter-org-files)
             (string-equal (directory-file-name (file-name-directory (buffer-file-name)))
                           (directory-file-name (expand-file-name dotcrafter-dotfiles-folder))))
    (message "Tangling %s..." (file-name-nondirectory buffer-file-name))
```

```
      (dotcrafter-tangle-org-file buffer-file-name)))

(defun dotcrafter--key (key)
  (kbd (concat dotcrafter-keymap-prefix " " key)))

(define-minor-mode dotcrafter-mode
  "Toggles global dotcrafter-mode."
  nil
  :global t
  :group 'dotfiles
  :lighter " dotcrafter"
  :keymap
  (list (cons (dotcrafter--key "t") #'dotcrafter-tangle-org-file)
        (cons (dotcrafter--key "u") #'dotcrafter-update-dotfiles))

  (if dotcrafter-mode
      (add-hook 'org-mode-hook #'dotcrafter--org-mode-hook)
    (remove-hook 'org-mode-hook #'dotcrafter--org-mode-hook)))
```

Things to try:

- Add a new output file to Desktop.org
- Edit `README.org`
- Turn off `dotcrafter-tangle-on-save`
- Turn off `dotcrafter-mode`

¶ What's next?

In the next episode, we'll learn how to write a custom major mode that provides a simple user interface for the `dotcrafter.el` package!

We're getting pretty close to having a package worthy of release!