# The FCM User Manual

The File Contents Manager, Version 1.0 beta 1

**Didier Verna** <didier@xemacs>

# Table of Contents

# Copying

# 1 Introduction

A great majority of the files that you open in a text editor such as `XEmacs` contain two kinds of material: text that only *you* can write (source code, composed natural language *etc.*) and markup that can largely be guessed, computed or automated (headers, footers, copyright notices, timestamps *etc.*)

`FCM`, the File Contents Manager, is an `XEmacs` library designed to help you with the latter kind of material. `FCM` works by associating the files you write with so-called *templates*. Templates not only provide initial, possibly dynamic contents for new files but are also used to establish a structural match with existing files, so that you can automatically update the templated contents at different times such as loading, saving, or upon explicit request.

Template files mostly contain plain text. When `FCM` encounters plain text in a template, it just copies it *as-is* to the file you're creating. Template files may also contain so-called *template variables* which allow you to manage contents dynamically. This comes in handy for inserting timestamps, copyright years, choosing a specific license *etc.* When `FCM` encounters a variable in a template file, it evaluates it in order to produce the actual contents to insert; a process known as *variable expansion*.

Template variables are said to be *automatic* when they expand to their value directly (as a timestamp for instance). Another kind of variable is the *interactive* ones. Interactive variables prompt you for some information everytime they need to be expanded (as a file tagline for instance).

`FCM` comes with a set of predefined templates but you can also create and use your own.

While its most frequent use is probably to provide initial contents for newly created files, `FCM` can act at other moments too. Apart from initialization, `FCM` will act at load time, save time and *update* time, that is, upon explicit request. By using so-called *variable modifiers*, template variables can be made to expand automatically at those different times, in addition to when a file is created. Finally, you can also explicitely request re-expansion of any variable at any time, hence dynamically updating the templated contents.

# 2 Installation

## 2.1 Distribution

FCM is available as a standard XEmacs package. You can get it in four different ways:

- Install it from within XEmacs via the package user interface (accessible from the 'Tools' menu or via *M-x list-packages*).
- Download the 'fcm-<version>-pkg.tar.gz' tarball from the XEmacs package repository and unpack it where you put the other packages.
  ftp://ftp.xemacs.org/pub/xemacs/packages/
- Get the sources from the XEmacs CVS repository. The module name is 'fcm'.
  http://www.xemacs.org/Develop/cvsaccess.html
- Get the sources directly from my website. You will also find several inlined versions of this documentation there.
  http://www.lrde.epita.fr/~didier/software/xemacs.php

If you're installing FCM for a whole site, like a company or a university, you might want to customize the default templates in order to make them suitable to a format that all employees should comply to (see Chapter 4 [Writing Template Files], page 9). As we will see in the next chapters, individual users may still use personal templates should they wish to do so.

## 2.2 Requirements

FCM currently works only with XEmacs 21.4 or later. I'm not sure it works with earlier versions of XEmacs, but I'm sure it does **not** work with GNU Emacs.

## 2.3 Getting Started

Once FCM is properly installed, you don't need any special trickery in your '.emacs' file to make it work, because all entry points to the library are autoloaded. You may however add an optional call to (fcm-install) in your '.emacs'. This will add a "Contents Management" submenu in the "Edit" menu giving you GUI access to FCM. From there, you can start using FCM without really knowing what's going on under the hood.

Contents management by FCM operates through a minor mode called fcm-mode. By switching this mode on and off, you can effectively toggle FCM on a per-file basis (FIXME: ref to force argument). fcm-mode is an autoloaded function which also installs the "Contents Management" submenu if needed. That is why the call to fcm-install in your '.emacs' is optional.

FCM can also be set to activate automatically on all qualifying buffers (see Chapter 3 [Buffer Qualification], page 7). This is done by switching on another minor mode called fcm-global-mode. This mode is *not* buffer-local. You can activate it in your '.emacs' but if you do, please do so as late as possible. fcm-global-mode is an autoloaded function which also installs the "Contents Management" submenu if needed. Consequently, there is no need for putting *both* a call to (fcm-install) and to (fcm-global-mode) in your '.emacs'.

Finally, every customizable FCM option can be accessed through the fcm custom group.

# 3 Buffer Qualification

In order for `FCM` to operate on a file properly, it needs to know which template is associated with the file and whether it should operate automatically at creation, load, or save time. The process by which `FCM` gathers this information is called *buffer qualification* ("buffer" because `FCM` obviously operates on file buffers). Whether `FCM` should operate at create, load, or save-time is called a *behavior*.

We said in the introduction that template variables may be set to automatically expand at those different times, so you may wonder why the additional information is required. In fact, it is not really required, but there may be times where you would rather skip the expansion process altogether, regardless of any template variable setting. In other words, the create, load, or save behaviors act as global switches.

## 3.1 Qualification Sources

`FCM` looks for behavior specifications at different places and in sequential order (see Section 3.2 [Qualification Process], page 8).

### 3.1.1 File Specific Behaviors

You can specify a set of behaviors for one particular file by setting the variable `fcm-behavior` in its local variables section. This variable's value must be of the form '`(:key` *`value`* `...)`' where the possible keys are `:template`, `:load` and `:save`.

`:load` and `:save` must be booleans that determine `FCM`'s behavior at the corresponding time, and `:template` must be a template file *designator* (see Section 3.3 [Template Designators], page 8).

As a safety precaution, every template file provided in the `FCM` distribution contains such a variable with a value of '`(:template nil)`', effectively disqualifying the file for contents management altogether.

### 3.1.2 File Name Based Behaviors

You can specify behaviors on a file name regular expression match basis by customizing the variable `fcm-behavior-files`. This variable must be a list of specifications of the form '`(regexp :key` *`value`* `...)`'. `regexp` is obviously a regular expression that will be use to match file names. The available keys are the same as before (see Section 3.1.1 [File Specific Behaviors], page 7) with the addition of the `:create` key.[1]

By default, `FCM` disqualifies files located in `/tmp/`.

### 3.1.3 Major Mode Based Behaviors

You can specify behaviors on a major-mode basis by customizing the variable `fcm-behavior-modes`. This variable must be a list of specifications of the form '`(mode :key` *`value`* `...)`'. `mode` is a symbol naming a major-mode (without the `-mode`) postfix. The available keys are the same as before (see Section 3.1.2 [File Name Based Behaviors], page 7)

By default, `FCM` disqualifies text, ChangeLog, outline and fundamental files.

### 3.1.4 Default Behaviors

Finally, you can also specify default behaviors by customizing the variable `fcm-behavior=defaults`. This variable must be of the form '`(:key` *`value`* `...)`' where the available keys are the same as before.

By default, the template designator is `"%m.fcm"` (see Section 3.3 [Template Designators], page 8) and `FCM` is set to perform at all times (create, load and save).

[1] Specifying a create behavior cannot be done within the file itself. Otherwise, it would already exist. . .

## 3.2 Qualification Process

When `FCM` needs to know the value for a particular behavior, it looks for a specification sequentially in the local variable `fcm-behavior`, in `fcm-behavior-files`, in `fcm-behavior-modes` and finally in `fcm-behavior-defaults`. `FCM` won't stop looking until all qualification sources are investigated. In particular, this means that when you provide only a partial qualification, the seearch will continue.

For example, suppose you have provided the following entry in `fcm-behavior-files`: `("/home/joe/" :load nil)`. This qualification is only partial because not all behaviors are specified. When `FCM` needs to figure out the save behavior for a file in your home tree, it won't stop there just because there is a filename match. The search will stop there in terms of filename match, but will continue with major-modes and defaults until an explicit setting for the `:save` key is found (or until all qualification sources are exhausted).

Because of the way this qualification process works, you also need to note that setting a behavior to `nil` explicitely is different from not setting it at all.

## 3.3 Template Designators

# 4 Writing Template Files

# Appendix A Indexes

## A.1 Concepts



## A.2 Variables



## A.3 Functions



## A.4 Keystrokes

(Index is nonexistent)