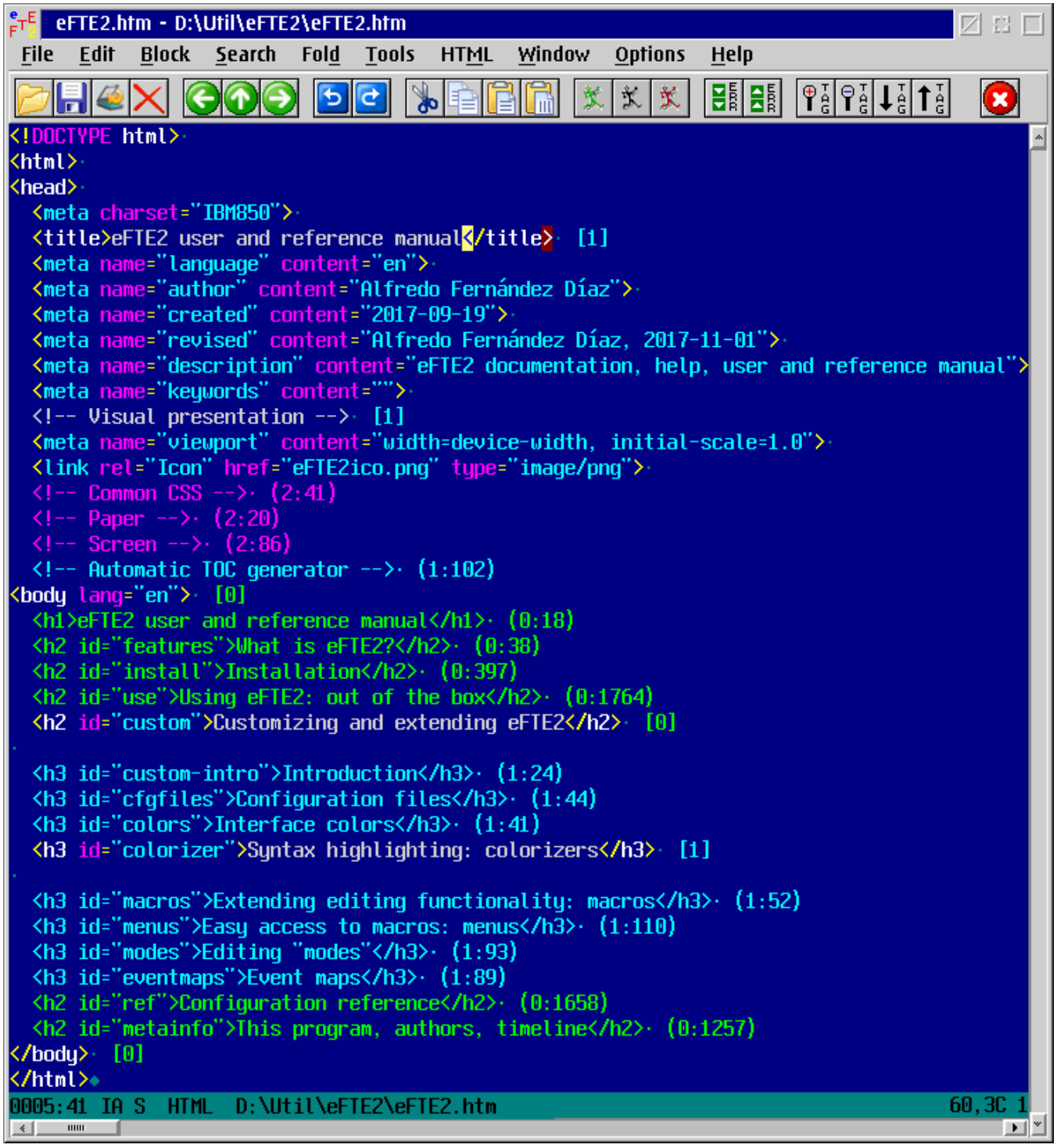
eFTE2 user and reference manual



By Alfredo Fernández Díaz
November, 2017
Last revision: 2017-11-01

Table of Contents

What is eFTE2?

eFTE2 is a free multiplatform text editor loaded with features that make it the editor of choice of many programmers and power users alike:

- It is a *FOLDING* text editor.
- Almost every major feature is customizable:
  - Editing modes for different file types (including C/C++, REXX, Perl, HTML, Java, etc).
  - State-machine-based syntax parsing and highlighting.
  - Pull-down menus tailored for the current file type.
  - Common and mode-specific shortcut keys (with nice CUA defaults).
  - Programmable editing macros, which can be linked to both shortcut keys and menus.
- Can operate on multiple files independently.
- Hypertext list of file "routines", as defined in each language.
- Supports ctags.
- Named bookmarks.
- Highlight all occurances of a word.
- Automatically highlights matching symbol when cursor is on any parenthesis, brackets, or braces. Great for nesting checks.
- Most stuff you might expect from any editor worth its salt: block indenting, entab/detab, case changes on text block, sort lines, column marking/copying, call to external programs (such as compiling), regular expression search and replace, screen splitting.
- Light and fast, can be used over a remote console / ssh connection (non-GUI versions anyway?).
- Multi-platform (currently OS/2, Win32, and Linux/X11).

Installation

This section is devoted to what component files of eFTE2 are supposed to get installed where.

Installation of eFTE2 should be a simple matter, such as unzip and run, which gives the end user full control (and responsability!) of which files are installed where, overwritten, etc.

However, eFTE2 is also distributed in alternate installer packages for managed installation on modern systems. In those cases, it is up to the user and system documentations to deal with system installers and their idiosyncrasies—we infer if you are reading this, you must have been able to install the application, or at least unpack it.

Compatibility with old FTE installs

If you want to install eFTE2 directly over an FTE/eFTE install on OS/2 and not lose your custom configuration changes you can achieve that by installing only packages 1 and 3 of the WPI to your current install directory.

In the future you should be able to use eFTE2 with the configuration files of an existing FTE install simply by creating a `mymain.fte` file in the same directory as your old `main.fte` and setting `EFTEDIR` to point there. Currently this is not possible in all cases because of an undocumented change introduced in eFTE: before it, `includes` in the configuration were interpreted as being relative to the file they were found in, but in eFTE this was changed to 'relative to main configuration'.

Example: say you have two configuration bits `a.fte` and `b.fte` under the directory 'menu', and the main configuration includes "menu/a.fte". FTE would include "`b`" from "`a`" with "`include b.fte`", but eFTE and current eFTE2 need it to be "`include menu/b.fte`".

This will be changed in the future to help preserving backwards compatibility in as many cases as possible.

Base files

The following files are included in all archives:

`README2`
: Readme file.

`edefault.fte`
: Fallback configuration file. The file `edefault.fte` should be kept in eFTE2's install directory. It provides a basic configuration primarily intended for editing files when the regular configuration is broken.

`efte2.hlp`
: The manual you're reading right now.

`config\*.fte`
: Configuration files.

`HISTORY`
: History of changes.

README.efte
efte readme file.

file_id.diz
Program description for BBS upload.

COPYING
GNU license.

Artistic
Artistic license.

AUTHORS
Contributor credits.

On OS/2 / eCS / ArcaOS

efte.exe
VIO console-only executable.

eftepm.exe
PM GUI executable.

Suggested installation file paths:

- %TOOLS%\eFTE\efte.exe
- %TOOLS%\eFTE\eftepm.exe
- %TOOLS%\eFTE\efte2.hlp
- %TOOLS%\eFTE\edefault.fte
- %TOOLS%\eFTE\config*.fte
- %TOOLS%\eFTE\docs\README.efte
- %TOOLS%\eFTE\docs\HISTORY
- %TOOLS%\eFTE\docs\README2
- %TOOLS%\eFTE\docs\COPYING
- %TOOLS%\eFTE\docs\Artistic
- %TOOLS%\eFTE\docs\AUTHORS

Place executable files somewhere on your `PATH`. The configuration files should be located in the config subdirectory. If you wish to install the config files somewhere else you may need to SET the EFTEDIR environmental variable to point to that directory. By default eFTE and eFTEPM look for the file `mymain.fte` as the root configuration file. All the rest of the configuration files are 'included' by earlier files. You can name your 'root' file something else but you will then need to use the `-c` switch followed by the full pathname for that file. For example:

```
efte -cx:\efte\config\mynewconfig.fte
```

To get the most out of eFTE2, you may install some additional programs and include various help files in your HELP and/or BOOKSHELF paths in config.sys. Even if you are using Open Watcom and don't

need the IBM OS/2 toolkit, you should install it and include the path to its help files in HELP. The same is true for your compiler help files. See the list of help files below. Most of these files come with OS/2, the OS/2 toolkit, Open Watcom or WarpIN. The few that don't are available on Hobbes. It is also recommendable putting your eFTE2 directory in your `PATH` and `DPATH` in CONFIG.SYS.

These are the programs eFTE2 is setup to work with if they are found in the PATH:

Open Watcom (http://www.openwatcom.org/)

Wmake and the C/C++ help files are available directly from the menus (Can easily be changed if you use something else).

See additional notes on using Open Watcom below.

Grep

Called directly from the menu; You can click on the results to open the file at the found line.

ISpell

Provides reasonable spell checking.

SVN and CVS

Direct access to the most commonly used commands for these versioning systems.

HTML Tidy

Direct access via menu.

Note on using Open Watcom: The 'Tools' menu entry 'Make and open list file' invokes directly 'wdis' via the totally undocumented command 'MakeListFile'.

The object file must be in the compile directory or directory relative to the compile directory (see below) f.e. ..\object\my.obj. Absolute paths are not currently supported.

Compile directory is the directory containing the file you are currently editing the first time you run/ compile/etc. After that the directory remains the same as long as the Messages buffer is left open. The directory is listed on the 'running' line [running 'myprogram ' in directory] Directory is the compile directory. One way to change it is to close the Messages buffer and run/ compile from a file in a different directory.

OS/2 developer documentation packages are linked to as well in the 'Help' menus of the appropriate editing modes. You will be able to directly consult the following books if they are available in your `HELP` path:

OS/2 Programming Guide

The following inf files make up this book: addendum.inf +CP1.inf +CP2.inf +CP3.inf +GPI1.inf +GPI2.inf+ GPI3.inf +GPI4.inf +MMREF1.inf +MMREF2.inf +MMREF3.inf +MMSSPG.inf +PM1.inf +PM2.inf +PM3.inf +PM4.inf +PM5.inf +WPS1.inf +WPS2.inf +WPS3.inf.

Watcom C Help

The following inf files make up this book: clib.inf +clr.inf.

Watcom C++ Help

The following inf files make up this book: cpplib.inf +wpperrs.inf.

EMX Help
The following inf files make up this book: emxbsd.inf +emxlib.inf +emxdev.inf +emxrt.inf.

IPF Help (ipfref.inf)
Make Help (Open Watcom tools.inf)
REXX Information (rexx.inf)
REXX Multimedia (mcirexx.inf)
REXX Tips and Tricks (rxtt36.inf)
TCP/IP REXX FTP API (rxftp.inf)
TCP/IP REXX Sockets API (rxsocket.inf)
WarpIN Programmer's Guide and Reference (wpi_prog.inf)

On Windows

On Linux

List of Configuration Files

This is a complete list of the config files that will be installed on a clean install. Updates are meant to be included in a file related to the file being update i.e. mym_html.fte for m_html.fte. In practice any one of them may get edited by the user. This means updating them will be difficult.

If you wish to modify only a specific mode, add customizations to a `my*.fte` file and 'include' it at the end of file. To also modify all of the descendants, 'include' customizations right after the include statement for mode.

Main/Global configuration

- main.fte
- mymain.fte
- myfontsize.fte
- systemmain.fte
- global.fte

User interface color definitions and schemes

- color.fte
- pal_b_kb.fte
- pal_base.fte
- pal_blk.fte
- pal_blue.fte
- pal_bluez.fte
- pal_gray.fte
- pal_nce.fte
- pal_wht.fte

HTML character sets and conversion tools selectable from menus (incomplete)

- charset\ents.fte
- charset\jap.fte
- charset\latin.fte
- htmlchar.fte
- htmlconv.fte

Mode specific text highlighting

- m_4gl.fte
- m_a51.fte
- m_ada.fte
- m_asm.fte
- m_asm370.fte
- m_basic.fte
- m_batch.fte
- m_bin.fte
- m_c.fte
- m_catbs.fte
- m_clario.fte
- m_cmake.fte
- m_cnfgs.fte
- m_css.fte
- m_diff.fte
- m_ebnf.fte
- m_eiffel.fte
- m_euphoria.fte
- m_falcon.fte
- m_fort90.fte
- m_fte.fte
- m_gawk.fte
- m_groovy.fte
- m_html.fte
- m_icon.fte
- m_idl.fte
- m_ipf.fte
- m_java.fte
- m_ldsgml.fte
- m_lisaac.fte
- m_lua.fte
- m_make.fte
- m_markup.fte
- m_merge.fte
- m_mod3.fte
- m_msg.fte
- m_mvsasm.fte

- m_ocaml.fte
- m_pascal.fte
- m_perl.fte
- m_php.fte
- m_plain.fte
- m_py.fte
- m_resdlg.fte
- m_rexx.fte
- m_rpm.fte
- m_rst.fte
- m_ruby.fte
- m_sgml.fte
- m_sh.fte
- m_siod.fte
- m_sl.fte
- m_sml.fte
- m_source.fte
- m_sql.fte
- m_tcl.fte
- m_trp.fte
- m_tex.fte
- m_texi.fte
- m_text.fte
- m_unrealscript.fte
- m_vhdl.fte
- m_wis.fte
- m_xml.fte
- m_xslt.fte

User interface styles

- ui_brief.fte
- ui_ew.fte
- ui_fte.fte
- ui_mixed.fte
- ui_ne.fte
- ui_ws.fte
- uicstyle.fte

Experimental (incomplete) expansions of the UI (included as a base for further development)

- Experimental\m_vi.fte
- Experimental\m_xp.fte
- Experimental\rgbcolor.fte
- Experimental\ui_k_joe.fte
- Experimental\ui_vi.fte

Mode-specific keyboard hotkey definitions

- kbd\k_c.fte
- kbd\k_fte.fte
- kbd\k_groovy.fte
- kbd\k_html.fte
- kbd\k_java.fte
- kbd\k_perl.fte
- kbd\k_rexx.fte
- kbd\k_rst.fte
- kbd\k_sgml.fte

Abbreviation expansion for various modes

- ab_c.fte
- ab_c_os2.fte
- ab_java.fte
- ab_perl.fte
- ab_rexx.fte
- ab_sh.fte

Default Menus (Normally English but can be any at user discretion)

- menu\m_c.fte
- menu\m_css.fte
- menu\m_groovy.fte
- menu\m_html.fte
- menu\m_html_t.fte
- menu\m_ipf.fte
- menu\m_make.fte
- menu\m_resdlg.fte
- menu\m_rexx.fte
- menu\m_rst.fte
- menu\m_sgml.fte
- menu\m_wis.fte
- menu\ui_k_brf.fte
- menu\ui_k_fte.fte
- menu\ui_k_ne.fte
- menu\ui_k_ws.fte
- menu\ui_m_ew.fte
- menu\ui_m_fte.fte
- menu\ui_m_ne.fte
- menu\ui_m_ws.fte

Internationalized menus where xx is the country code (optional)

- menu_xx\m_c.fte
- menu_xx\m_css.fte
- menu_xx\m_groovy.fte
- menu_xx\m_html.fte
- menu_xx\m_html_t.fte
- menu_xx\m_ipf.fte
- menu_xx\m_make.fte
- menu_xx\m_rexx.fte
- menu_xx\m_rst.fte
- menu_xx\m_sgml.fte
- menu_xx\ui_k_brf.fte
- menu_xx\ui_k_fte.fte
- menu_xx\ui_k_ne.fte
- menu_xx\ui_k_ws.fte
- menu_xx\ui_m_ew.fte
- menu_xx\ui_m_fte.fte
- menu_xx\ui_m_ne.fte
- menu_xx\ui_m_ws.fte

Using eFTE2: out of the box

Manual conventions

Along this document, the syntax of program command lines, eFTE2 arguments and macro editing commands, and others will be explained using a uniform selection of symbols, compiled here for your convenience. They are:

- When not assigned another specific meaning (such as in sections regular expression match operators, or keybindings) square brackets [and] indicate optional arguments or text.
- When not assigned another specific meaning, angle brackets < and > indicate an example that must be adjusted to particular cases, without the brackets.
- When not assigned another specific meaning, or appearing literally in code, curly braces { and } denote a set of mutually exclusive values.

Also, a number of operations can be performed according to conditions expressed independently of each other by the presence or absence of some characters in a so called 'flags' string. Such characters are referred to as 'flags' for short, and flags strings are used in settings as well as some interaction with the user.

Command line options

The command line syntax is:

```
efte[pm] [[options] [file(s)]]
```

If no options nor file(s) are specified, eFTE2 will start in file browsing mode.

Options

`-lline[,column]`
Go to line (and column) in next file specified on command line.

`-m[MODE]`
Use mode `MODE` for remaining files. If no argument is specified, mode override is cancelled.

`-C or -![file]`
Use specified configuration file (full pathname). If no argument is specified, the default configuration `edefault.fte` is used.

`-D[file.dsk]`
Load/save desktop from `file.dsk`. If no argument is specified, desktop loading/saving is disabled.

`-H[file.his]`
Load/save history from `file.his`. If no argument, disable history load/save.

`-Ttags`
Load tags file `tags`. The file must be in the format generated by the *ctags* program.

-ttag
Lookup tag named `tag` and display file containing it.

--
The rest of the arguments are not options, but filenames.

-+
The next argument is not an option even if starting with a '-'.

--help -h or -?
This shows a usage dialog.

--debug / --debugclean
These start logging; `debug` appends efte.log, `debugclean` creates a new one.

--version
Provides some version information.

N.B.: there should not be any delimiter between an option and its arguments.

Examples

efte[pm] -mBIN efte.exe
load efte.exe in BIN mode.

efte[pm] -l100,30 win.c
go to (100,30) in win.c.

efte[pm] -! -l100,30 mymain.fte
use the default configuration and go to (100,30) in mymain.fte.

efte[pm] window.cpp
load file window.cpp.

efte[pm] -dGNU window.cpp
load file window.cpp using the GNU indent style.

efte[pm] --debug window.cpp
load file window.cpp with logging enabled in append mode.

efte[pm] -mBIN efte.exe -m window.cpp
load efte[pm].exe in binary mode, window.cpp in default mode (C/C++).

efte[pm] -mBIN -+ -bla-
load file -bla- in BIN mode.

efte[pm] -- -1 -2 -3 -4 -5 -6
load files -1, -2, -3, -4, -5, -6.

efte[pm] -D -H efte.dsk efte.his

Disable desktop and history loading and saving and load files efte.dsk and efte.his.

Under OS/2, NT and DOS default history and desktop files are named `eFTE.DSK` and `eFTE.HIS` respectively. Under Unix they are named `.efte-desktop` and `.efte-history`. The global desktop and history files will be searched in program directory under OS/2 and in user home directory under Unix.

Standard 'plain' text editing

eFTE2 default configuration follows the common user access (CUA) guidelines, so it should be rather intuitive how to perform typical text editing tasks and get started. However, the capabilities of eFTE2 extend beyond those of basic system text editors, such as E, NotePad, or the like, so it is a good idea to explore them in some depth. Furthermore, basic text editing tasks can be daisy-chained in "macros", a feature which is obviously easier to take advantage of if at least a superficial grasp of the basic editor capabilities has been acquired.

Text vs Plain mode

There are two basic styles of editing text files, stream and line editing. Stream editors treat files as one long stream of characters, line editors treat files as a series of individual lines, separated by a line end character. Being originally oriented towards programming, eFTE2 is feature-rich on the line editing side, but since both editing styles are valid interpretations of the same reality, and stream editing is so popular with word processors and many other applications, eFTE2's set of features lets the user work seamlessly either way. (eFTE2 also features another variation: 'column-' editing.)

The difference, however, may be very important regarding specific files. In general, computer code and other text formats where line numbers and exact text position are important tend to have many short lines to help clarity and avoid lateral scrolling in editors. On the other hand, text documents tend to have paragraphs condensed in single, [very] long lines, precisely because line numbers and exact text position are not considered important in them. Text documents are thus generally expected to have their lines automatically rearranged, split, and wrapped by word processors or other applications as necessary for comfortable editing.

By default, 'plain' files are never reflowed in eFTE2, while in 'text' files word wrap is set to automatically divide lines as the right margin is reached when writing. Word wrap can also be set to continuously reflow any text block delimited by empty lines (a "paragraph") as it is edited.

A word of caution: there is always 'undo' and 'discard changes', but at the very least, check the status line when you're starting to edit files with eFTE2—'plain' and 'text' modes are *not* the same thing!

Status line

eFTE2 always shows a status line at the bottom of its screen.

While editing a file, the status line shows the following information:

`curpos|flags|mode|mod?filename[ ... ]curchar|winno`

Where each field represents:

cursor position

line:column

flags

I
Insert

A
Autoindent

C
Matches are case sensitive

SLC
Stream, Line, or Column block-mode

wW
Automatic word wrap active (w = line, W = paragraph).

mode
Mode name as specified in configuration file.

mod?
***** if file was modified, **%** if file is read-only.

curchar
Decimal ASCII code of character under cursor, or EOL/EOF.

winno
'window' number

Routines, buffers, or directory browser screens display lists of selectable items, with a final status line at the bottom showing the item count and the index of the current selection within it on the right corner; the rest of the information displayed should be self-explanatory.

Keys, menus and tasks

Most tasks in eFTE2 should be accessible with one or two keystrokes, but eFTE2 is built with customizability in mind, starting with key assignment. It would then be a bit self-defeating to give a command key reference when key shortcuts may well be used for different tasks in different interface configurations or editing modes. A task reference is given instead.

Unlike other text editors, eFTE2 is built with accessibility and clarity in mind as well, so all basic tasks are bound to pull-down menus (also accessible through the mouse), which in turn are organized in logical groups.

As menus unfold, you may notice some characters are underscored or highlighted in a different color; by pressing the corresponding keys you can activate the right menu entries significantly faster than getting to them with the cursor keys.

The default configuration menus also reflect what key combinations (or shortcuts) each task is bound to, so it should be easy to learn by heart the key combinations in each mode by opening the menus a few times, and then effectively not needing them any more.

Further input required

Menu items linked to tasks that require any kind of further user input for proper execution typically have an ellipsis (…) in them. You can see this in "Open...", "Save as...", and other menu items, and also in some entries pointing to sub-menus.

Submenu entries are always marked with →, so those marked with an ellipsis too indicate that all actions linked to the submenu entries require additional user input as well.

The tasks that will be described now are general-purpose, and as such they should be available in all editing modes—in most cases they correspond to single internal editing commands of the editor, and so their actions are not described directly, but are linked instead to the appropriate section of the commands reference in most cases.

Once a file is opened in eFTE2, the tasks laid out under the application menus are described in the following sections.

File

Under the File menu, you find pretty much the same options as not just in every other text editor, but pretty much in every other application in existence: load a file, save changes, exit the application, etc., with possibly only a couple of exceptions: the reference to "editing modes", and "Next/Previous".

It is often overlooked that eFTE2 is capable of editing several files at once: the humble "Next/Previous" labels stand for "Next/Previous *file*" (or, more accurately, buffer), each of which will be edited in the appropriate *mode*.

Entry	Key (if any)	Command / action
Open...	F3	FileOpen
Open in Mode...	Ctrl+F3	Opens menu for picking the editing mode before opening a file.
Open Directory	Ctrl+M	DirOpen
Reload	Shift+F3	FileReload
Save	F2	FileSave
Save As...	Shift+F2	FileSaveAs
Save All	Ctrl+F2	FileSaveAll
Write To...		FileWriteTo
Print		FilePrint
Next	Alt+Right	FileNext
Previous	Alt+Left	FilePrev
Close	Alt+Q	FileClose
Close All		FileCloseAll
Exit	Alt+X	ExitEditor

Open in Mode

The entry "Open in Mode..." of the menu "File" offers to open a file from a selection of submenu entries. The difference with regular "Open file..." is that the editing mode for the file is established first through

the appropriate menu entry. Then the file can be opened normally, be it by specifying its name, or from the file browser.

It is out of the scope of this manual to explain what all the programming / scripting / markup languages formats, and their associated modes are. However, 'plain' or 'text' editing modes are very important to understand, for the other modes are directly derived from one of these.

Edit

Entry	Key (if any)	Command / action
Undo	Alt+BackSp	Undo
Redo	Alt+Shift+BackSp	Redo
Cut	Shift+Del	BlockCut
Copy	Ctrl+Ins	BlockCopy
Cut-Append		BlockCutAppend
Copy-Append		BlockCopyAppend
Paste Stream	Shift+Ins	BlockPasteStream
Paste Column	Alt+Ins	BlockPasteColumn
Paste Line		BlockPasteLine
Clear	Ctrl+Del	BlockKill
Line		Opens submenu Line
Quote Literal...	Ctrl+Q	InsertChar
ASCII Table...	Ctrl+Sh+A	ASCIITable

Line

Entry	Key (if any)	Command / macro
Insert line	Shift+Enter	LineInsert
Add line	Alt+Enter	LineAdd
Split line	Ctrl+Enter	LineSplit
Join line	Ctrl+J	LineJoin
Duplicate line	Ctrl+D	LineDuplicate
Delete line	Ctrl+Y	KillLine
Center line		LineCenter
Delete to line end	Alt+End	KillToLineEnd
Delete to line start	Ctrl+Sh+BackSp	KillToLineStart
Comment	Ctrl+Alt+C	MoveLineStart; ?FindReplace /^(\s*)/ /\/\/ \1/ "xnc"; MoveDown
Uncomment	Ctrl+Alt+U	MoveLineStart; ?FindReplace /^[\/][\/] / // "xnc"; MoveDown
Uppercase		LineCaseUp
Lowercase		LineCaseDown
Togglecase		LineCaseToggle

Entry	Key (if any)	Command / macro
Rot13		LineTrans 'A-Za-z' 'N-ZA-Mn-za-m'
User specified...		LineTrans

You may have noted that this commenting lines out / back in is in the style of C programming. This behaviour should be expected to be adapted to what is considered a comment in different languages.

Block

Entry	Key (if any)	Command / action
Unmark	Esc	BlockUnmark
Mark Stream	Alt+A	BlockMarkStream
Mark Column	Alt+K	BlockMarkColumn
Mark Line	Alt+L	BlockMarkLine
Select Word		BlockSelectWord
Select Line		BlockSelectLine
Write...		BlockWrite
Read Stream...		BlockReadStream
Read Column...		BlockReadColumn
Read Line...		BlockReadLine
Print		BlockPrint
Indent	Alt+I	BlockIndent
Unindent	Alt+U	BlockUnindent
ReIndent	Alt+\	BlockReIndent
Translate		Opens submenu Translate.
Expand Tabs		BlockUnTab
Generate Tabs		BlockEnTab
Sort		BlockSort
Sort Reverse		BlockSortReverse

Translate

Entry	Command / macro
Uppercase	BlockCaseUp
Lowercase	BlockCaseDown
Togglecase	BlockCaseToggle
Rot13	BlockTrans 'A-Za-z' 'N-ZA-Mn-za-m'
User specified...	BlockTrans

Search

Entry	Key (if any)	Command / action
Find...	Ctrl+F	Find
Find Next	Ctrl+G	FindRepeat
Find Prev	Ctrl+H	FindRepeatReverse
Find and Replace...	Ctrl+R	FindReplace
Place Bookmark...		PlaceBookmark
Goto Bookmark...		GotoBookmark
Tags		Opens submenu Tags
Match Parenthesis	Alt+-	MatchBracket
Goto Line...	Alt+J	MoveToLine
Goto Column...		MoveToColumn
Current Word		Opens submenu SearchWords

SearchWords

Entry	Key (if any)	Command
Search Prev	Alt+,	SearchWordPrev
Search Next	Alt+.	SearchWordNext
Highlight	Alt+/	HilitWord

Tags

"Tags" are a way of creating automated hypertext links between, for example, function calls and function definitions (even across files!).

For this to work with the file(s) currently being edited, another index (or "tag") file must have been generated first by CTags (see http://ctags.sourceforge.net/) or an equivalent program.

From the CTags manual:

> Ctags generates an index (or "tag") file of names found in a set of files in a number of languages. Depending on the language, functions, variables, class members, macros and so on may be indexed. This tag file allows these items to be quickly and easily located by a text editor or other utility. A "tag" signifies a language object for which an index entry is available (or, alternatively, the index entry created for that object).
>
> Alternatively, ctags can generate a cross reference file which lists, in human readable form, information about the various source objects found in a set of language files.

Typically, you would want to know if something in one file is referenced in others. Once loaded in the editor, the tags file acts as a proxy allowing to jump between all references in the 'regular' files to the items listed in the tags. Some additional examples could be:

1. We are editing the editor configuration and we want to check if matching `colorize` and `eventmaps` are defined. Or, some menu imports as submenu another menu from another file. The names of these objects would be in the tags file.
2. Elements of an HTML document with `class` or `id` attributes could have these used as "." or "#" selectors in a CSS file. The attribute values would be in the tags file.

Entry	Key (if any)	Command
Find word	Ctrl+]	TagFindWord
Search tag...	Ctrl+Sh+]	TagFind
Go back	Ctrl+[	TagPop
Next tag	Alt+]	TagNext
Previous tag	Alt+[	TagPrev
Load tags...		TagLoad
Clear tags		TagClear

An expected difficulty when trying to use the CTags feature is lack of support for more modern languages. This should not be a problem because CTags was developed with flexibility and extensibility in mind, but sometimes it is for users because the documentation is not all that clear about how to add that support themselves.

Support for new languages must be added to CTags via command line parameters, or adding these to its configuration file (`.ctags`). This is an editor-centric example of support for the eFTE2 config 'language' as must be added to the CTags configuration:

```
--langdef=fte
--langmap=fte:.fte
--regex-fte=/^[ \t]*menu[ \t]+(.*)[ \t]+/\1/u,usermenu,UI menu definitions/
--regex-fte=/^[ \t]*colorize[ \t]+([A-Za-z0-9_-]+)([ \t]*:[ \t]*([A-Za-z0-9_-]+)[ \t*])/\1 \3/c,colorize,syntax highlight schemes/
--regex-fte=/^[ \t]*eventmap[ \t]+([A-Za-z0-9_-]+)([ \t]*:[ \t]*([A-Za-z0-9_-]+)[ \t*])/\1 \3/e,eventmap,menu and key bindings/
--regex-fte=/^[ \t]*mode[ \t]+([A-Za-z0-9_-]+)([ \t]*:[ \t]*([A-Za-z0-9_-]+)[ \t*])/\1 \3/m,mode,editing modes/
```

Please note that the syntax used for CTags regular expressions is a bit different from the one eFTE2 uses.

Fold

The idea behind folding is that the text you are editing is often composed of conceptual blocks that you want to be reminded of, but don't necessarily want to have to scroll through. For example, a programmer might want to see all the methods of a class on screen at once, but not necessarily the whole implementation of each method (and if you did show the implementation, it would not all fit on one screen). But even more than programming code, one thing that benefits *hugely* from folding is HTML. Here is an example, based on the HTML source of this manual:

```
<h4>SaveFolds</h4>  (0:10)
<h4>Colorizer</h4>  [0]
    Specifies a previously declared <A
    HREF="#colorizer">colorize</A> mode to use for syntax
    highlighting in this editing mode.
<h4>Loading files in various formats</h4> (0:43)
```

The first and last line here are color-highlighted within eFTE2 itself. The parenthesis (also color-highlighted to stand out) after the folded lines quickly tell you that, e.g., the "SaveFolds" section has 10 lines in it, and is a top-level fold (it might contain sub-folds). It takes just a couple keystrokes to fold away "Colorizer" and display the "SaveFolds" body.

If you happen to need to look at the same file in a different text editor, the folds are indicated by comments appropriate to the language being used, for example:

```
<h4>Loading files in various formats</h4> <!--fold00-->
```

It adds a few extra characters to the file, but doesn't interfere with its actual function outside of this editor.

Once the folding concept is illustrated, the 'folding' tasks referenced below should not be difficult to understand:

Entry	Key (if any)	Command
Open fold	Ctrl+Gr+	FoldOpen
Open nested folds	Ctrl+Gr*	FoldOpenNested
Open all folds	Alt+Gr*	FoldOpenAll
Close fold	Ctrl+Gr-	FoldClose
Close all folds	Alt+Gr/	FoldCloseAll
Create fold	Alt+Gr+	FoldCreate
Create folds by regexp...		FoldCreateByRegexp
Create folds at routines		FoldCreateAtRoutines
Destroy fold	Alt+Gr-	FoldDestroy
Destroy all folds		FoldDestroyAll
Promote	Sh+Gr-	FoldPromote
Demote	Sh+Gr+	FoldDemote
Toggle	Ctrl+Gr/	FoldToggleOpenClose

Options

The "Options" menu entries let the users change settings that affect the behaviour of eFTE2 while editing the file.

Entry	Key (if any)	Command / macro / action
Change mode [more]		Opens a submenu to change the editing mode for current file.
Change C indent style		Opens a submenu to choose from a selection of popular styles.
Toggle		Opens submenu Toggle

Entry	Key (if any)	Command / macro / action
Insert mode	C+O C+I	ToggleInsert
Word wrap	C+O C+W	ToggleWordWrap
Left margin...	C+O A+[	ChangeLeftMargin
Right margin...	C+O A+]	ChangeRightMargin
Show markers	C+O C+.	ToggleShowMarkers; WinRefresh
Highlight tags		ToggleHilitTags; WinRefresh
Show bookmarks		ToggleShowBookmarks; WinRefresh
Show tabs	C+O Tab	ToggleShowTabs; WinRefresh
Tab size...	C+O C+T	ChangeTabSize; WinRefresh
Expand tabs	C+O C+Tab	ToggleExpandTabs; WinRefresh
Insert tabulator	S+Tab	InsertTab
File Trim EOL		FileTrim; WinRefresh
Indent block	Alt+\	BlockReIndent; FileTrim; WinRefresh
Print to...		SetPrintDevice

Toggle

Entry	Key (if any)	Command
Character case	A+`	CharCaseToggle
Auto indent	C+O C+A	ToggleAutoIndent
Case sensitive	C+O C+C	ToggleMatchCase
Trim EOL spaces	C+O C+E	ToggleTrim
Undo/Redo	C+O C+U	ToggleUndo
Read only	C+O C+R	ToggleReadOnly
Keep backups		ToggleKeepBackups
Make backups		ToggleMakeBackups
Backspace Unindents		ToggleBackSpUnindents
Indent with tabs		ToggleIndentWithTabs
Space tabs		ToggleSpaceTabs
Backspace kill tab		ToggleBackSpKillTab
Delete kill tab		ToggleDeleteKillTab

Local (pop-up) editing menu

This is an internally called 'Local' menu, not another entry in the main application menu. It pops up while editing a file when you click with the 2nd mouse button on the background of the editor window, or alternatively if you press its key shortcut (Shift+F10 by default).

It gives access to the most frequently used actions relevant to the current file, line, or selected block of text, all of which have been covered already:

Entry	Key	Command / macro
Unmark	Esc	BlockUnmark
Cut	Shift+Del	BlockCut
Copy	Ctrl+Ins	BlockCopy
Paste	Shift+Ins	BlockPasteStream
Paste Column	Alt+Ins	BlockPasteColumn
Delete line	Ctrl+Y	KillLine
Delete to EOL	Alt+End	KillToLineEnd
Indent block	Alt+\	BlockReIndent; FileTrim; WinRefresh
Save	F2	FileSave
Close	Alt+Q	FileClose

The contents of this menu are customizable as well; note that different pop-up menus are available in the other views of eFTE2 (see Non-editing tasks).

Non-editing tasks

No tasks under certain application submenus have been covered so far—eFTE2 lets the user do other tasks that, being handy to have covered from within the editor, and very useful at that, are not part of the editing itself, so they have their own separate section.

Window

Items under the menu Window let the user decide what information eFTE2 displays, and how it is visually presented. The task reference and some term definitions follow:

Entry	Key (if any)	Command / action
New Frame		FrameNew
Split Horizontal	Ctrl+F4	WinHSplit
Close view	Shift+Alt+F4	WinClose
Close other views	F5	WinZoom
Next view	F4	WinNext
Prev view	Shift+F4	WinPrev
Load Desktop		DesktopLoad
Save Desktop		DesktopSave
Save Desktop As		DesktopSaveAs
Routines	Ctrl+I	ListRoutines
Buffers	Alt+0	ViewBuffers
Directory	C+M	DirOpen

Frame

In GUI environments, a new program window executing eFTE2. Frames can be closed one by one, or all at once.

View

Each eFTE2 window (whether text mode- or a GUI session) may be split in sub-displays stacked vertically, named 'views', only one of which can have active focus at a time to receive user input. All of them act like independent editor windows (syntax highlight, key bindings, status line) within the editor display, and the active view controls the common visual elements of the display, f.e. contents of the application menu.

Note that different views are not the same thing as different buffers (see note about Views vs. buffersin the next section).

Desktop

A special text file where eFTE2 stores a list of editing buffers, position in them, etc. Following the same metaphore as popular operating systems, this provides a quick way to replicate previous work sessions by opening again the same files and directories that were open at some point in time, like returning to a real desktop to continue what was being done before leaving it.

Routines
Buffers
Directory

Routines: parts of a file

While editing a file, you can open an alternate view (called a buffer) that shows just a summary of the meaningful units of your text (function definitions, HTML headings, class declarations, etc.), one per line. Highlight any unit from this list and press enter to jump there.

The concept is similar to folds, but operates independently of what is folded. Because of the similarity, when editing a file eFTE2 actually lets you define file folds based on the routines found in it.

What text constructs the editor will consider as "routines" exactly depends on the file type being edited, and thus on the corresponding editing mode. In the configuration settings for that mode, routines must be defined using a regular expression that allows the editor to find them.

The routines buffer does not allow for editing, it only allows to select routines to move quickly through long files, so its menus have few entries, and others are very simplified compared to those shown when editing the file.

Buffers

In eFTE2 terminology, a 'buffer' is any channel used to store information for the user to select and browse or otherwise work on what is in it. The buffers list will show open files, file browser instances, message queues, or the EventMapView buffer if open, and can be used as a quick way to switch between them, or do some rudimentary managing of open files (limited to saving or closing them).

The buffers list is in itself a buffer, and is thus listed at the top, because it is the only one that cannot be closed.

Unless files are opened in different instances of the editor, operating systems cannot show them independently in their tasks lists, so this is the only way to list all files currently opened in it.

It is possible to switch from one editing buffer to the previous or next one using the File menu, or jump direclty to one of the first ten using the default key combination Alt+#, where # is the buffer number displayed for each buffer in the list.

Notes:

1. Different views can show the same buffer, for example a file being edited. They do not edit the file independently, though, but only display independently parts of it: if the same [region of a] buffer is shown in more than one view, any action that takes place in the active one is immediately reflected in the other(s).
2. Different editing buffers can be opened for the same file (for example by re-opening it via the file browser), though. If this is the case, these can be shown and act totally independently in different views.

Directory

eFTE2 incorporates a minimal file browser which provides the ability to make directories, navigate the file system, and rename or delete individual files. (See "Navigate" menu below.)

The file browser lists, in its own buffer(s), the contents of the last directories examined when manually picking files for editing.

"Navigate" menu

Entry	Key (if any)	Command
Reload	Ctrl+R	Rescan
Go < level	Ctrl+PgUp	DirGoUp
Go > level	Ctrl+PgDn	DirGoDown
Go to \	Ctrl+\	DirGoRoot
/ Goto Dir...	/	DirGoto
Rename File		RenameFile
Make Directory		MakeDirectory
Delete File	Ctrl+D	DeleteFile

Help

Entry	Key (if any)	Command / macro
Contents	F1	ShowHelp "eFTE2" ""
Keyboard	Alt+F1	ViewModeMap
Show key		ShowKey
About...		ShowVersion

On OS/2 systems only (?).

EventMapView

From pretty much anywhere in eFTE2, selecting 'keyboard' from the 'Help' menu, or pressing the corresponding key combination (Alt+F1 by default) will bring up this unique buffer listing keyboard bindings.

EventMapView list starts with the mode last active when it was invoked, then each item in the list shows a key combination plus a dump of the command macro that it will execute. The listing will recursively repeat the listing for all parent modes linked by inheritance.

Editing files to be used with other programs

It is of course possible to write general-purpose text files, or all your relevant documents in plain text, but you will be most likely using eFTE2 to edit text files following some particular style or structure so the resulting file will be used by another application lacking the ability of editing such files with the same flexibility and freedom, be it because that is not its primary function, or whatever other reason. Examples of this could be:

- Writing documents to pass on to a typesetting system such as TeX.
- The sources to some computer program written in programming languages like C/C++ before it is compiled.
- An HTML document to be published online and viewed in internet browsers.
- An interpreted program, e.g. a REXX or Perl script.
- Online manuals or help for application programs, with internal links and different contexts for help or references.

In all of these cases and many more, the final use of a file with an application other than the text editor is the culmination of an often lengthy task that could benefit from intermediate stages such as spell-checking, testing internal references, suitability for compilation, etc. eFTE2 provides the ability not only to invoke external applications to execute such tasks, but also to capture and process their output for further use, if relevant, within eFTE2.

For example, a compiler will write to the standard output to inform you of mistakes needing your attention at some locations in a program source file before it can be further used. If executed from within eFTE2, such compiler output could be used to open the file in eFTE2 at those precise locations for immediate action to take place.

Because FTE was initially conceived as a programming editor, the editing command and setting that govern this behavior in eFTE2 in each editing mode are actually called Compile and CompileRx. However, this feature can be useful in any process for which one or more suitable analogs for the compilation / check compiler messages for feedback exist. In fact, most entries in the standard* 'Tools' pull-down menu do some variant of (some part of) it.

* specific editing modes may execute a substantially different set of tasks from their 'Tools' menu.

Typically, a file must be saved immediately prior to invoking a tool, as will be done with the standard ones, so any modifications the tool might make to the file will not conflict with those made in the editor since the last time the file was saved.

Output messages from external applications

There are two basic internal commands to run an external program from eFTE2: `RunProgram` and `Compile`.

When `RunProgram` is used, programs are executed without special regarding to any possible output. GUI eFTE2 runs programs aynchronously in a separate session, but the text mode version runs them as a

child process. In this case, it is still possible to read at least the last part of any output by pressing Alt+F5*, like in the old Borland IDEs.

* This is the default key combination. For reference, the internal command executed is `ShowEntryScreen`.

On the other hand, when an external program is executed via `Compile`, its output is captured by eFTE2 and put into yet another buffer. With the aid of an adequate regular expression to parse it, the Messages buffer will let you move quickly through the application output and jump from one error to another skipping the rest, or return to the exact position in the file that was 'compiled', if that information is available.

This is the local (pop-up) menu task reference for messages buffers:

Entry	Key (if any)	Command / action
View error	Enter	Activate
Previous error	F11	CompilePrevError
Next error	F12	CompileNextError

Tools: standard external programs

By default eFTE2 is ready to execute a number of standard tools available on most platforms, and suitable for a variety of tasks:

Entry	Key (if any)	Command / action
Compile...	F9	Compile "wmake -e "
Grep...		Compile "grep -n -I "
Make and Load List File *		MakeListFile
Save and ISpell		FileSave; RunProgram "ispell.exe ".$FilePath; FileReload
Shell	Alt+F9	RunProgram ""
Run...	Ctrl+F9	RunProgram
Previous error	F11	CompilePrevError
Next error	F12	CompileNextError
Messages	S+F9	ViewMessages
Clear Messages		ClearMessages
CVS	C+O_C+V	Opens submenu CVS
SVN	C+O_C+N	Opens submenu SVN

* On OS/2 systems only (?). See Notes on using Open Watcom, under installation.

Grep…

grep is a standard command-line utility for searching plain-text files for lines that match a regular expression. Its name comes from the "ed" editor command g/re/p ("globally search a regular expression and print").

See https://en.wikipedia.org/wiki/Grep for more information.

ISpell

Ispell is a program, part of the GNU system, that helps you to correct spelling and typographical errors in a file. When presented with a word that is not in the dictionary, ispell attempts to find near misses that might include the word you meant.

Supposedly ISpell may be superseded by a similar tool called ASpell, for which support might be added in the future.

Version control in concurrent environments

Both CVS and SVN are applications for controlling the changes made / applied to files in a concurrent work environment. With slightly different approaches, CVS and SVN let you compare a set of files in a central repository with a local copy, and selectively synchronize either by applying individual changes or sets of them in the appropriate direction.

While primarily intended for cooperative software development, these systems can be applied, at least in theory, to any environment meeting the aforementioned conditions.

Detailed use of either versioning system is beyond the intended scope of this manual, but under the pull-down menu 'Tools', you can see two sub-menus that give direct access to the most frequently used commands of both and their output from within eFTE2 with some more specialized macros/internal commands:

CVS

Entry	Command / action
CVS Check	`RunCvs "-n update"`
CVS Update	`RunCvs "update -d"`
CVS Diff	`CvsDiff ""`
CVS Commit	`RunCvsCommit ""`
CVS Add	`RunCvs "add"`
CVS Remove	`RunCvs "remove"`
CVS Status	`RunCvs "status -v"`
CVS	`Cvs`
View CVS	`ViewCvs`
View CVS Diff	`ViewCvsDiff`
View CVS og	`ViewCvsLog`
Clear CVS messages	`ClearCvsMessages`

SVN

Entry	Command
SVN Status	RunSvn "status"
SVN Update	RunSvn "update"

Entry	Command
SVN Diff	SvnDiff ""
SVN Commit	RunSvnCommit ""
SVN Add	RunSvn "add"
SVN Remove	RunSvn "remove"
SVN Log	RunSvn "log"
SVN Revert	RunSvn "revert"
SVN Blame	RunSvn "blame"
SVN	Svn
View SVN	ViewSvn
View SVN Diff	ViewSvnDiff
View SVN log	ViewSvnLog
Clear SVN messages	ClearSvnMessages

Regular expressions

Introduction

While most developers might be expected to get acquainted with regular expressions at some point, this is not necessarily the case with many 'regular' users.

Regular expressions, used extensively within eFTE2, are bits of text that often look a lot like file filters with 'wildcards' in them, think f.e. *.txt, or *12?.htm?. They also provide an advanced method of representing or searching and replacing text in your files, much more powerful than working with wildcards or static text strings.

In most old text editors, and even many word processors, users are not expected to do anything significantly beyond inputting text, and maybe simple substitution stuff like replacing 'cat' with 'dog', or perhaps ' ,' with ', ' for the typographically-inclined. Search / replace operations like "find full stops not followed by upper case letters" or "replace any sequence of more than one space with a single one" are quite often hopelessly impossible except using specific, 'advanced' functions or tools present in few programs.

Regular expressions use a relatively simple syntax to encode expressions similar to the aforementioned ones in machine-processable form. In most regular expression 'dialects' those would look very similar to:

`.*\.\l`
: "any character string followed by a dot immediately followed by a lowercase letter, no spaces in between".

`\s+`
: "a sequence of at least one blank character (space, tab, newline or carriage return)".

Formally, regular expressions consist of normal characters, and characters with a special meaning, called 'operators'. Operators allow you to anchor matches, match classes of characters, match a given pattern several times or match among alternate patterns. Operators can be also used to group simple patterns to form more complex ones. (See regular expression match operators for reference.)

Regular expressions in action

To illustrate how regular expression matching works, an example might come in handy.

You might want to search for tags with a named anchor inside in an HTML file, like `<h4><a name="destination">A header</a></h4>`.

Depending on how much details we need to care about within our targeted tags, this regular expression might suffice:

```
\<(\w+)\>\<a name="([^"]*)"\>[^\<]*\<\/a\>\<\/\w+\>
```

Borrowing some knowledge from the regular expression syntax reference, let's examine a bit more in detail what this expression matches:

`\<(\w+)\>`

\w+ stands for "one or more word characters" (A to Z and a to z plus numbers in most cases). Being between parenthesis, whatever it matches must be registered for later use, and it must also be enclosed in angled brackets "<" and ">", so that expression bit efectively means "find and remember any single word between < and >", or in HTML terms, "any opening tag, without attributes".

Notes:

1. w is escaped so it means "word character" instead of being taken as a regular letter; on the other hand, both "<" and ">" have to be escaped to be taken literally and not as regular expression operators.
2. Only the tag name is registered, without the angled brackets— these are *outside* the parenthesis.

`\<a name="([^"]*)"\>`

This is again an `a` tag with the attribute `name` being assigned some value between double quotes. The value, `[^"]*`, can thus be any string of zero or more (the `*` bit) characters other than the double quote (`^"`). Whatever is inside the quotes is registered for future reference as well.

`[^\<]*`

Any number of characters (again even none at all) other than "<" (escaped with "\"). This goes on until a new HTML tag opening character "<" or the next one in the expression are found in the text.

`\<\/a\>\<\/\w+\>`

The closing tag "", followed by *any* other closing tag. Note that in reasonably well-formed HTML this would match the first tag, but this covers mismatched tags as well.

So this expression will match any HTML string following that precise pattern. Some examples might be:

1. `<h1><a name="start">Title</a></h1>`
2. `<h2><a name="section_n">Some intermediate header</a></h1>`
3. `<h3><a name="please_fill_in"></a></h3>`
4. `<p><a name="note_on_xyz">This might be a footnote.</a></p>`

In general, it is best to start matching text with simple regular expressions that match text quite rigidly and gradually incorporate elements that may or may not be present in the text to be matched, to make

them more flexible and effective: for example, the matching expression above could account for extra tag attributes, or spaces around them too.

After a pattern is thus matched to a regular expression, it can be replaced with a simple text string or another pattern with its own operators. (See regular expression replacement operators for reference.)

In the example above, we might want to modernize the HTML code, and replace the `name` attribute of the anchor tag with an `id` attribute for the opening tag, keeping the same value, and get rid of the anchor altogether. Using the same expression as before, the following one could be a suitable replacement: `<\1 id="\2">\3<\/\1>` (note the same tag is used twice now to ensure opening and closing tags of the element do match).

The above would replace the previous matched tags examples with:

1. `<h1 id="start">Title</h1>`
2. `<h2 id="section_n">Some intermediate header</h2>`
3. `<h3 id="please_fill_in"></h3>`
4. `<p id="note_on_xyz">This might be a footnote.</p>`

N.B.: If you inspect closely example #2, you will see how both opening and closing tags now do match.

Regular expressions in eFTE2

Regular expressions can be typed in eFTE2 when doing Find / Replace operations, but they are also used in many parts of the configuration, be it to define editing mode file names, the text constructs that make up the proper routines in that particular mode, ways to analyze output of other programs, or as part of macro commands involving Find / Replace actions, for example in the alternate HTML mode 'Tools' menu. Naturally, the same syntax and operators are used everywhere; see the regular expressions reference for details.

Editing modes (example: HTML)

This section of the manual is intended to illustrate the kind of enhanced editing you can expect from mature specialized editing modes, part of the decision-making involved in writing some parts, and how using them can make your life easier. HTML mode will be used as an example, leaving out as many details about HTML itself as possible.

This new editing mode builds up on plain mode, and thus shares all of its functionality and menus, but also extends them by adding some automated editing functions written with the HTML format peculiarities in mind.

These functions are accessible through hotkeys and two application menu entries, corresponding to two main categories: a new one simply called 'HTML', and 'Tools', where a number of submenus have been changed or added.

For this manual, it will suffice to consider HTML as regular text interspersed with special constructs called 'entities' and 'tags' (not to be confused with search tags). Regular text is mostly unpredictable in its input patterns, and already covered by the standard editing functions anyway, so enhancements to HTML editing are aimed directly at dealing with these special elements.

New colors schemes

Different types of text (programming code, tagged text) may have specific syntax rules that benefit greatly from coloring.

HTML tags follow a rigid syntax, so one thing that definitely helps to write them right is coloring, or syntax highlighting, i.e. when editing an HTML file, different colors are used to write all parts of a tag so you can not only spot them against the normal text they are interspersed with, but also so you can tell whether they are well formed or not at a glance.

HTML tags are written like this:

```
<tag attribute="value"[ more attributes... ]>text</tag>
```

So in eFTE2 you should see the symbols <, >, / and = in the code punctuation color when they are part of a tag, and different colors for 'tag', 'attribute', 'value' and 'text'.

Text colors are changed and assigned at different positions according to the sequence of characters found prior to getting there. F.e., in HTML specifically the colorizer is programmed so that '<' triggers the change from default color to punctuation, then to 'tag', and '>' gets back to 'normal' from 'tag'. Also, while in 'tag' sub-mode, 'attribute' is entered automatically at the first space, and '=' and quotes change from 'attribute' to 'value', etc.

Now, the colorizer goes a little yet important step beyond that: you are not supposed to make up your tag or attribute names, but to pick them from pre-defined lists. Being a mere text-editor eFTE2 will of course let you type anything you want, but at the same time, if you use words which are not in the lists for tags or attributes, yet another color is used so you can tell on the spot whether your code matches what is programmed into the editing mode. It can even color a few 'forbidden' keywords from other lists.

Now, the colorizer goes a little yet important step beyond that: you are not supposed to make up your tag or attribute names, but to pick them from pre-defined lists. Being a mere text-editor, eFTE2 will of course let you type anything you want while, at the same time, it will try to match your input against word lists. This lets several colors be used so you can tell on the fly whether your input matches any 'allowed' (or 'forbidden'!) keywords on different spots.

Examples:

- `<MyTag>You are not supposed to do this</MyTag>`
- `<p MyAttribute="do not do this either">More made-up stuff</p>`
- `<p style="text-align: center;">Now this is legal</p>`
- `<center>Was deprecated years ago</center>`

New editing functions

Every entry under the menu "HTML" is devoted to speed up writing HTML tags and entities. These can be classified in several groups according to their complexity, but all items within the same group are equivalent for all practical editing purposes, so only a working description of each group will be given.

HTML entities, colors, single tags

A first approach could be automatic insertion of text strings that are hard to memorize, or long to type, for example HTML entities. That is exactly what you will find under the submenus "Colors" or "Special characters". No more trying to remember whether 'black' is '#000000' or something completely different.

Just put the cursor where you want it, select the right menu, and the editor will write it in for you. Ditto for math symbols or what have you.

Tag pairs

The next level in complexity stems from most HTML tags coming in opening and closing pairs, f.e.: `<i>this should be cursive</i>`.

An easy try at this could be to have the editor to write both tags at once, f.e. "`<i></i>`". Short as this is (two or three keystrokes vs. typing both these shortest tags), it wouldn't be quite practical: HTML tag pairs enclose text between them most of the time, so any carefully written macro should take this into account.

And indeed, the method currently employed to write HTML tag pairs does that precisely: it will move to the start of any block of text selected, write an opening tag, then move to the end of the block, and write the closig tag; if no text is selected at all upon invocation, the whole process falls back to writing both tags at cursor position.

If you try it, you may wonder why such tag pair injections leave any text selected previously plus the newly injected tags selected upon termination. The reason is simple: doing so provides an easier way to serialize these operations. Imagine you need to enclose a text block in two tag pairs. If the first enclosing ended and unselected the block, it would be necessary to select it again for the second one, and only you (the user) can do that, so you are required to do it—by hand. On the other hand, if the block is left selected, the second enclosing can be executed directly. But what if no second operation were to be performed, you may ask? Well, unmarking the block can be done with just an additional keystroke, or, and this is the good part, with an additional *automated* operation.

Tags that need to be at a specific location

Some HTML tags need to be at specific locations; for example, DTD tags must be at the very top of the file, and encoding tags need to be in the HTML head block, but before any other tag within it, especially the document title, to avoid getting funny-looking characters.

If you were editing a file and suddenly you remembered you need to include one of these tags, you would need to search for the appropriate location to insert them, and then type them in (or do it from one of those menus intended to save you precious time). But the necessary location is marked, so moving the cursor there can be done as well with an automated search!

And that is exactly what all entries under 'Base tags' or 'Document head' do: they insert the required tag(s) only after making sure a suitable location is found in the file, and moving the cursor there.

Putting it all together

As with all good building block sets, the ability to automatically write tags at cursor position, or at specific locations, is pretty much all that is needed to create a more complex structure, such as a new HTML document (empty but for the necessary tags) from scratch—it is just a matter of doing all of those things sequentially, so that would be the next logical step in automatisation.

But can more interesting things be done? after all, it is even easier to have empty document templates in store than creating them from scratch every time, no mater if it takes only a few keystrokes. It would be really much nicer to have the possibility of combining HTML templates with existing text documents.

Said and done. We could enclose existing text in HTML tags, couldn't we? So, let's just open an ASCII text file in HTML mode, select its whole contents, and an HTML template can be built around it before

going on with the editing! It is nearly the same thing as the multiple tag inclusions discussed in the previous section.

And a new whole submenu under 'HTML' is born: 'Create structures' (from text).

New and tailored tools

Besides adding tags to text faster than simply typing them, there is the question of dealing with HTML already written in a document opened for editing. All tags start with "<", and end with ">", so they are easy to find for doing some nifty tricks!:

Zap tags

First thing would be getting rid of tags quickly. Under "Tools" you will find a "Zap HTML tags" submenu:

< >? Current <? >

Assumes cursor is inside a tag (between "<" and ">"), and deletes the whole tag. If cursor is between two tags, they both and any text between them will be erased.

Examples:

- `<!-- this whole comment [cursor] will be erased -->`
- `<img class="undesired image" [cursor] ... src="...">`
- `<p>This whole paragraph [cursor] will be erased.</p>`

Previous

Looks up first tag ending anywhere before the current cursor position, and deletes it. Cursor does not even need to be moved near the tag.

Example:

`<p class="tag to be deleted">Some text [cursor] … </p>`

Next

Looks up any tag starting anywhere to right of cursor, and deletes it. (The closing `</p>` in the example above.)

Both

Performs both previous operations.

Example:

`"<p>Some [cursor] text.</p>" → "Some [cursor] text."`

Duplicate tags

If for some reason existing HTML tags must be duplicated, the tools described above are replicated under "Duplicate HTML tags", but they will copy and paste again the tags instead of deleting them.

Elements: text between tags

A simple definition for an HTML element could be "plain text enclosed between matching opening and closing tags," for example:

```
<p>A simple text paragraph.</p>
```

Some obvious operations can be greatly accelerated with the right specialized tools:

Complete closing tag

This will look for any tag starting before the cursor, and will add a matching closing tag at the end of the line the cursor is in.

Example:

```
<p>This paragraph will have a closing tag added...
[cursor] somewhere.<!-- Here -->
```

Complete opening tag

This will look for any ending tag starting after the cursor, and will add a matching opening tag at the beginning of the line the cursor is in.

Example:

```
<!-- Opening tag will be created here -->This paragraph wasn't prop-
erly started [cursor], but it can be—
as long as this closing tag is found:</p>
```

Split at cursor

If the cursor is placed before an element closing tag, it can be split in two at cursor by replicating the closing tag. Example:

```
<p>Some text.[cursor]Some more text.</p>
```

will be split into:

```
<p>Some text.</p>[cursor]<p>Some more text</p>
```

Merge with next

Sometimes, two adjacent elements must be merged, which is equivalent to deleting the end tag of the first, the open tag of the next, and the space in between. Example:

```
<p>This text[cursor]</p>
<p>should be connected</p>
```

will be merged like this:

```
<p>This text[cursor]should be connected</p>
```

Please note these 'element' operations are not aware of the Document Object Model, or HTML well-formedness, and they are not a magic wand, just some basic tools oriented at speeding up manual editing of reasonably clean HTML code. They are easy to trip up if you are not careful!

Metadata extraction

It is quite frequent to find HTML documents which contents have been merely pasted on some template system, so the document will look OK at first glance but soon enough you'll start spotting details like the document title being, well, 'document title'. A couple of extra tools have been included to automate doing some search, copy and text replacement to palliate common cases of such sloppy publishing.

Under the submenu "Extract metadata" you will find:

Title
: Looks for the first `h1` element available in the document, copies its contents, and replaces any document title element with that.

Author
: Looks for an element similar to `<p class="author">Name here</p>`, and adds a `meta` tag to the file replicating that information.

Extended search and replace

Recurrent search and replace operations in HTML can be too tedious and complicated to type them in every time, so they are best automated too. All of the following can be performed from the corresponding menus in any selected block of text:

- Make empty elements (non-)self-closing, f.e.: `<br>` ↔ `<br />`.
- Transform non-ASCII characters to HTML entities and viceversa. (According to selected encodings.)
- Convert named entites to numeric, and viceversa.
- Convert ASCII constructs to HTML, like simplified entitites (f.e. "`(c)`" for "©"), line breaks, paragraphs, etc.

HTML Tidy (external application)

Specialized HTML editing macros and all, when editing HTML it is easy to make mistakes. From the Tidy website:

> Wouldn't it be nice if there was a simple way to fix these mistakes automatically and tidy up sloppy editing into nicely layed out markup? HTML TIDY is a free utility for doing just that. It also works great on the atrociously hard to read markup generated by specialized HTML editors and conversion tools, and can help you identify where you need to pay further attention on making your pages more accessible to people with disabilities.
>
> Tidy is able to fix up a wide range of problems and to bring to your attention things that you need to work on yourself. Each item found is listed with the line number and column so that you can see where the problem lies in your markup. Tidy won't generate a cleaned up version when

there are problems that it can't be sure of how to handle. These are logged as "errors" rather than "warnings".

Tidy will be invoked directly from the 'Tools' after choosing the current HTML file encoding from its submenu. For this to work, Tidy must be available in your `PATH` (or its equivalent in Unix systems).

Alternate editing modes included

Currently, eFTE2 incorporates all these mode definitions in its configuration. This listing was generated from a tags file.

Mode name	Definition file	Inherits from mode	Additional information
GL	m_4gl.fte	SOURCE	
ASM51	m_a51.fte	SOURCE	
Ada	m_ada.fte	SOURCE	
ASM	m_asm.fte	SOURCE	
ASM370	m_asm370.fte	SOURCE	
BASIC	m_basic.fte	SOURCE	http://www.freebasic.net
Batch	m_batch.fte	PLAIN	DOS .BAT and JP Software Command Processor BTM files, by Michael DeBusk
BIN	m_bin.fte	PLAIN	Binary mode
C	m_c.fte	SOURCE	
CATBS	m_catbs.fte	PLAIN	For viewing nroff output (do NOT use for editing)
CLARION	m_clario.fte	SOURCE	
CMAKE	m_cmake.fte	PLAIN	
CNFGSYS	m_cnfgs.fte	PLAIN	DOS and OS/2 CONFIG.SYS files, by Michael DeBusk
CSS	m_css.fte	PLAIN	Cascading Style Sheets (*.CSS) files to dress up HTML.
DIFF	m_diff.fte	PLAIN	
EBNF	m_ebnf.fte	PLAIN	
Eiffel	m_eiffel.fte	SOURCE	
EUPHORIA	m_euphoria.fte	SOURCE	http://openeuphoria.org/
FALCON	m_falcon.fte	SOURCE	
FORTRAN	m_fort90.fte	SOURCE	Fortran-90
FTE	m_fte.fte	SOURCE	The editor configuration 'language'.
GAWK	m_gawk.fte	SOURCE	
GROOVY	m_groovy.fte	SOURCE	
HTML	m_html.fte	PLAIN	Internet sites are woven HyperText Markup Language.

Mode name	Definition file	Inherits from mode	Additional information
ICON	m_icon.fte	SOURCE	https://www2.cs.arizona.edu/icon/index.htm
IDL	m_idl.fte	C	
IPF	m_ipf.fte	PLAIN	Text source format for IBM Information Presentation Facility binary online help and manuals (INF/HLP).
JAVA	m_java.fte	SOURCE	C without pointers?
LDSGML	m_ldsgml.fte	MARKUP	LinuxDoc SGML
LISAAC	m_lisaac.fte	SOURCE	http://isaacproject.u-strasbg.fr
Lua	m_lua.fte	SOURCE	
MAKE	m_make.fte	PLAIN	
MARKUP	m_markup.fte	PLAIN	The theory: all markup modes should inherit from this mode, which enables the user to easily modify preferences for all markup file types.
MERGE	m_merge.fte	PLAIN	
MODULA3	m_mod3.fte	SOURCE	Modula-3
MSG	m_msg.fte	TEXT	EMail messages
MVSASM	m_mvsasm.fte	SOURCE	
OCAML	m_ocaml.fte	PLAIN	
PASCAL	m_pascal.fte	SOURCE	
PERL	m_perl.fte	SOURCE	
PHP	m_php.fte	SOURCE	A popular server-side HTML preprocessing programming language.
PYTHON	m_py.fte	SOURCE	
RESOURCE	m_resdlg.fte	PLAIN	OS/2 and Win* dialog resource files.
REXX	m_rexx.fte	SOURCE	
NETREXX	m_rexx.fte	REXX	
RPM	m_rpm.fte	SOURCE	RPM spec files
reST	m_rst.fte	PLAIN	ReStucturedText files, http://docutils.sourceforge.net/
Ruby	m_ruby.fte	SOURCE	
SGML	m_sgml.fte	MARKUP	
SH	m_sh.fte	SOURCE	
SIOD	m_siod.fte	SOURCE	
sl	m_sl.fte	SOURCE	SLang
SML	m_sml.fte	SOURCE	
SOURCE	m_source.fte	PLAIN	The theory: All source modes should inherit from this mode which enables the user to easily modify

Mode name	Definition file	Inherits from mode	Additional information
			preferences for all source code file types.
SQL	m_sql.fte	SOURCE	
TCL	m_tcl.fte	SOURCE	
TEX	m_tex.fte	MARKUP	
TEXINFO	m_texi.fte	MARKUP	
TEXT	m_text.fte	PLAIN	
TRP	m_trp.fte	SOURCE	Trap reports generated by Exceptq/MapXQS.
UNREALSCRIPT	m_unrealscript.fte	C	
VHDL	m_vhdl.fte	SOURCE	
WIS	m_wis.fte	PLAIN	WarpIN installer script
XML	m_xml.fte	MARKUP	
XSLT	m_xslt.fte	HTML	

Customizing and extending eFTE2

Every major user interface feature in eFTE2 can be customized to fit your preferences or extend its functionality by editing the configuration, which is directly loaded from .FTE files, like `edefault.fte`. This file should be located and kept in the installation directory of eFTE2 as a failsafe mechanism:

Since the configuration is plain text, it can be edited from within eFTE2 itself. However, it is not possible to reload the configuration on the fly (yet?), and any syntax errors in the configuration will prevent eFTE2 from opening until they are fixed, so it is recommendable to leave eFTE2 open after editing and open a second instance to test the changes. If it fails you can easily edit/back out the changes in the first instance.

Another possibility is to make a copy of the current configuration and edit that instead. eFTE2 can switch between configurations using the `-c` command line switch followed by the full path to the alternate configuration.

If everything fails, use the "efte -!…" string from the error message. This will open the configuration at the problem line while using the default configuration.

Configuration files

To make management of complex configurations easier, the eFTE2 configuration is by default broken down into several smaller modules (modes, menus, keys) that are linked together via `include` directives in them. However, all configuration parameters of linked separate files are incorporated to the same single configuration space, which is referred to in this manual as 'the configuration (file)' for short.

On startup, the editor will attempt to load the configuration starting from `mymain.fte` in any of several standard places including:

- The eFTE2 install directory (can be named anything)
- efte\config
- efte\local

and under the "Program Files" and "HOME" directories from your environment. If you want the configuration files somewhere else or if eFTE2 is having trouble finding `mymain.fte`, you can use the environment variable `EFTEDIR` with `SET EFTEDIR = <MYPATH>`. eFTE2 will look in that directory and in a `config` and/or `local` subdirectory. If you have several sets of configuration files use `SET EFTEDIR` in a script to switch between them.

Four 'compiler directives' are defined to conditionally include or exclude configuration files. The directives are `%if`, `%endif`, `%define` and `%undefine`. They behave as expected and `!`, the "not" character, can be used in the defines. Do not try to put comments (or anything else) on these lines, as this will result in a syntax error. White space is tolerated. In the last section of the manual there is a complete reference of the configuration sections and syntax.

Note that the `include` directives allow different configurations to share important, non-changing bits.

In `mymain.fte` you can select the global UI style and color scheme as explained in the following section, or you can skip it and go to the main configuration file, `main.fte`. There you will find the global configuration settings, and includes to start linking editing modes and menus.

Interface colors

eFTE2 user interface colors are defined in `pal_base.fte` and grouped together in different color schemes in other FTE configuration files (`pal_*.fte`), which are selectively included in the main configuration file to allow for easy customization.

You can select any pre-made color scheme editing the file `mymain.fte`, edit the color schemes themselves, or create your own to be included anywhere in the configuration.

Color symbolic names and UI editor objects (f.e scroll bars) are linked to one another in `color.fte`.

Except for the possibility of rendering text or other objects unreadable or invisible with the same color for both foreground and background, changing the interface colors is independent of the rest of eFTE2 functionality.

The syntax for color settings is:

`color_palette` `{ { 'name', 'value' } [, …] }`

where 'name' is any symbolic name string defined for later use in any other parts of the configuration dealing with color, and 'value' is either a colon-separated pair of symbolic names as previously defined, or a space-separated couple of numeric values, corresponding to the standard PC color numbering in text video modes; the first item in 'value' is for the background color, the second one is for the foreground.

In the default configuration, `pal_base.fte` maps numeric colors to symbolic names like this:

```
{ 'black', '0 0' }
```

which are then used in `pal_*.fte` color schemes like this:

```
{ 'Editor_Selected', 'black:darkCyan' }
```

All numeric color values are listed in the color reference.

Text and colors: syntax highlighting

It would make very little sense to define different colors for text as seen in `pal*.fte` files if all text were to be rendered in the same one. eFTE2 lets the user to define sets of rules governing how text will change color as special symbols, words, or combinations of them are found, and form patterns that make edition easier for the user.

This is done creating a `colorizer` in the configuration. The safest way to try this is to create a new file, say `test.fte`, and then *optionally* include it from the main configuration files with "`oinclude test.fte`". This way, if this file breaks the configuration, it will suffice to rename or delete it and the editor will start as normal when it is not found.

Coloring files

The absolute minimum to do text highlighting would obviously be two colors. To do things a bit more interesting, four are defined here for internal use (assuming they actually map to four different colors as defined in the `pal*.fte`). This is a working skeleton `test.fte` file to expand:

```
colorize TEST {
    SyntaxParser = 'SIMPLE';
    color {
        { 'Normal',      'Editor_Default' },
        { 'Number',      'Lang_DecimalNumber' },
        { 'Punctuation', 'Lang_Punctuation' },
```

```
        { 'Comment',     'Lang_Comment' },
    };
}
mode TEST {
    FileNameRx = /\.\c{TST}$/;
    HilitOn    = 1;
    Colorizer  = 'TEST';
}
```

This will make eFTE2 open `*.tst` files in 'test' mode, all text in one color. Note the `colorize` is not bound in itself to file extensions, nor the other way round. This allows for the same colorizer to be used for different file extensions if necessary.

Changing colors

To alternate between colors, rules must be specified in the `colorize` section, after the `color` pairings between 'internal' colors just declared for highlighting and global color categories defined for the editor in `pal*.fte`. The rules for color change correspond to transitions in a 'state machine'. The first state must be declared like this:

```
colorize TEST {
    SyntaxParser = 'SIMPLE';
    color { [...] };
    h_state 0 { 'Normal' }
}
```

which should make all text appear in 'Normal' color, or any of the other three if the rule is changed accordingly.

Let us imagine now that `test` files allow for comments, ignoring all text appearing between the character '#' and the end of each line. This will require a second state and transitions from state 0 to 1 and back to be declared like this:

```
colorize TEST { [...]
    h_state 0 { 'Normal' }
    h_trans { 1, '', '#', 'Comment' }
    h_state 1 { 'Comment' }
    h_trans { 0, '$', '', 'Normal' }
}
```

which should result in lines of `test` files being split in two color regions on either side of wherever the character '#' appears.

Now some more color could be applied, for example, to mathematical operations:

```
colorize TEST { [...]
    h_state 0 { 'Normal' }
    k_trans { 0, 's', '-+:=;<>', 'Punctuation' }
    h_trans { 0, 'x', '[0-9]+', 'Number' }
```

```
    h_trans { 1, '', '#', 'Comment' }
    h_state 1 { 'Comment' }
    h_trans { 0, '$', '', 'Normal' }
}
```

Note that the two new highlight rules operate completely within state 0, so no coloring of numbers or operators takes place in the comments, to the right of '#'s. An inverted coloring pattern could be easily defined on the comments side like this:

```
colorize TEST { [...]
    h_state 0 { 'Normal' }
    h_trans { 0, 's', '-+:=;<>', 'Punctuation' }
    h_trans { 0, 'x', '[0-9]+', 'Number' }
    h_trans { 1, '', '#', 'Comment' }
    h_state 1 { 'Comment' }
    h_trans { 1, 's', '-+:=;<>', 'Normal' }
    h_trans { 1, 'x', '[0-9]+', 'Punctuation' }
    h_trans { 0, '$', '', 'Normal' }
}
```

Additionally, rules for highlighting specific keywords can be added to each state, either within a given state or with the possibility of state transitions when keywords are matched or not. The mechanism, though, remains essentially the same: as the editor explores the file contents, all characters found are checked against states and transition rules to be given a corresponding color. If the file contents are altered, so may the colors downstream be.

Please note that all transition rules for any given state are tested sequentially, that is, the moment a token in the file matches a transition rule, no more of them are tested, and the cursor is advanced in the file to start checking again the rules against the new token. It is generally a good idea to put more specific rules ahead of more general ones in order to reduce the likelihood of interference.

Syntax highlighting and text coloring can get complicated, and ten or more states are not a rarity among common colorizers; however, most cases can be adequately and briefly dealt with after the basic transitions outlined above are properly mastered.

Extending editing functionality: macros

Macros are sequences of actions executed automatically to speed up text editing. Once defined, macros are activated either by pressing an appropriate key combination or through interface menus, or when invoked from other macros.

Example: putting quotes around some text is done by moving the cursor to the beginning of the text, inserting a quote character ("), and doing the same again after moving the cursor to the end of the text to be quoted. This could be automated by defining a macro to execute after selecting some text.

In the configuration, macros are semicolon-separated series of commands between curly braces, i.e.:

```
{
    <command>; # (f.e. InsertString "Hello World!";)
    […;]
    <command;>
```

```
}
```

Any number of commands in the sequence can be in the same line; except for the last one, all of them must be followed by a semicolon.

The example mentioned above could thus look like this:

```
{ MoveBlockStart; InsertString '"';
MoveBlockEnd; InsertString '"' }
```

Commands in a macro can be optionally preceded by a whole number and a colon to be executed more than once (f.e.: 3:MoveDown), and a question mark if failing should not halt the macro (f.e.: ?Replace_1; ?Replace_2; ?Replace_3 …, or 2:?Find '/').

Macro execution will end after the last command is executed, or halt when the first command not marked with '?' fails.

Example — two command sets:

a. `Find 'place'; InsertString "X";`
b. `?Find 'place'; InsertString "X";`

These behave the same if 'place' is found ? cursor is moved there, and an X marks the spot.

OTOH if 'place' is not found, a) stops right there, whereas b) goes on and most likely the X ends up in the wrong place…

Easy access to macros: menus

The first way a user might want to invoke an editing macro, being unfamiliar with the editor, would be selecting it from a menu. Let us assume, then, that you want to use the example macro that was described before to put quotes around blocks of text.

Now, a natural place to put something like this might be the 'Block' menu, and that must be done at a specific location: `ui_m_fte.fte`, or whatever other UI style linked from `main.fte`—that one is simply the default.

The code for the menu block will look very similar to the following, maybe with different key shortcut labels:

```
menu Block {
    item "&Unmark\tEsc"          { BlockUnmark }
    item "Mark &Stream\tAlt+A"   { BlockMarkStream }
    item "Mark &Column\tAlt+K"   { BlockMarkColumn }
    item "Mark &Line\tAlt+L"     { BlockMarkLine }
    item;
    item "Select Wor&d"          { BlockSelectWord }
    item "Selec&t Line"          { BlockSelectLine }
    item;
    item "&Write…"               { BlockWrite }
    item "&Read Stream…"         { BlockReadStream }
    item "Re&ad Column…"         { BlockReadColumn }
    item "Rea&d Line…"           { BlockReadLine }
```

```
    item "&Print"               { BlockPrint }
    item;
    item "&Indent\tAlt+I"       { BlockIndent }
    item "U&nindent\tAlt+U"     { BlockUnindent }
    item "R&eIndent\tAlt+\"     { BlockReIndent }
    item;
    submenu "Translat&e",         Translate;
    item "E&xpand Tabs"         { BlockUnTab }
    item "&Generate Tabs"       { BlockEnTab }
    item "Sor&t"                { BlockSort }
    item "Sort Re&verse"        { BlockSortReverse }
}
```

Please note that shortcut keys are not really defined in the menu entries—if they exist, mere courtesy to the user dictates they ought to be reflected there. This might not be necessary in the future.

All you need to include in the code above is these lines before the closing curly brace (well, the first line is not really necessary):

```
menu Block {
    ...
    item;
    item "Put block in quotes"
      { MoveBlockStart; InsertString '"'; MoveBlockEnd; InsertString '"' }
}
```

The next time the editor is started, the item "Put block in quotes" will be the last one in the 'Block' menu, ready to go when selected after selecting some block of text.

But it would be even better to be able to invoke any user defined macros with their own shortcut keys, wouldn't it? After all, the purpose of having macros is to save time and every keystroke counts.

Instead of binding the same series of commands again to some key definition we will get to, the first real step to saving work would be to name the macro so it can be invoked just like any other command—from the menu above (or any other), via some key combination, or even from other macros.

This should be easy enough:

```
sub BlockQuote {
    SavePos;
    MoveBlockStart; InsertString '"';
    MoveBlockEnd; InsertString '"' ;
    MoveSavedPos;
}
```

Actually, commands in any macro can be either internal commands (such as MoveRight) or other macros that have been previously defined: any macro meant to be invoked by others must be declared before them.

Example: if we intend to have a macro like

```
sub ComplexStuff {
```

```
    [command(s);]
    SimpleStuff;
    [more command(s)]
}
```

where "SimpleStuff" is another macro we have created, SimpleStuff must appear in the configuration before ComplexStuff.

So, where in the configuration must named macros appear exactly?

If you have a look at `main.fte`, you will see that a bunch of "mode" (m_*.fte) files are included together. If you browse the bigger ones, you will see they include named macros in their bodies, but these are not enclosed in other sections. This means macros are global to the configuration, so putting them in one file or the other is just a matter of helping to manage complex configurations more easily.

Since the merging of contents into configuration space is done sequentially, care must be taken so files containing macro definitions that invoke other macros are included in the right order, but that is all there is to it. In general, though, putting macros for one "mode" into that m_*.fte file is enough—it is unlikely that macros conceived for some specific purpose are to be invoked from more than one mode.

But what is a "mode", anyway?

Editing "modes"

A **mode** is a collection of basic editor settings (such as tab size, or what is considered a comment) plus editing macros, menus, syntax highlighting, and/or sets of key combinations that are bound together to ease editing of specific file types—all of these settings are available at the same time when editing files of given types, thus conforming an 'editing mode' for them. Modes are declared like this in the configuration:

```
mode new[:old] { mode settings }
```

The mode named `new` inherits its initial settings from `old` parent mode if one is specified at declaration, then overrules any values with its own.

Loading files in various formats

Here are appropriate settings for loading files in various formats:

DOS/Win/OS2/NT text files (CR/LF delimited):

```
    StripChar 13
    LineChar 10
    AddCR 1
    AddLF 1
```

Unix text files (LF delimited):

```
    StripChar -1
    LineChar 10
    AddCR 0
    AddLF 1
```

MAC text files (CR delimited):

```
StripChar -1
LineChar 13
AddCR 1
AddLF 0
```

Binary files (fixed record length):

```
StripChar -1
LineChar -1
AddCR 0
AddLF 0
LoadMargin 64
ForceNewLine 0
```

Mode selection

To determine what mode to use for a file, eFTE2 will first check if such mode has been established by the command FileOpenInMode or a command line option (**-m**).

If no mode has been set, the file name is then matched to all FileNameRx defined in mode declarations.

If a mode has not been determined like that yet, the first line of the file will be read (up to 80 chars), to try and match it with the FirstLineRx declarations of all modes.

If all of the above fails, the editor will use the mode specified by the global setting `DefaultModeName` to load a file. If such mode does not exist, the first mode defined in the configuration will be used.

Mode, syntax highlight and event maps

Besides colors for syntax highlighting, the most visible effects in the editor come from "event maps". Remember, macros, syntax highlighting and menus are defined globally so they can be shared and linked to different editing modes when/where appropriate. `eventmap` sections of the configuration are used to bind user events (key presses and activation of menus) to one another and to specific modes.

Both syntax highlighting schemes and user events are linked to a given mode either by having in the configuration a corresponding configuration section of the appropriate type with the same name, i.e.:

```
mode 'MYMODE'[:PARENT] { mode settings }
colorize 'MYMODE'[:PARENT] { eventmap settings }
eventmap 'MYMODE'[:PARENT] { eventmap settings }
```

or by establishing the values `EventMap` or `Colorizer`in the mode settings.

This allows for similar modes to share `eventmap` and/or `colorize` definitions without the need to have additional sections that just inherit settings from a previous one.

Contributor's note: this also seems to be a fail-safe mechanism, as inferred from some notes in the configuration (notably `ui_vi.fte` file) which read "make sure proper eventmap is used".

Event maps

Eventmap settings sections of the configuration may include activation of menus, keybindings, and abbreviations as well. If <PARENT> if specified in the definition, any <PARENT> settings will be inherited by <MYMODE>.

The references for establishing the mode menus and keybindings for a mode should be self-explanatory.

Key combinations can be bound to any macro, just like menu items. Again, all commands in the macro are executed in sequence until one of them fails.

Keybindings are inherited from parent modes and optionally overriden.

Key names can be preceded by modifiers **A, C, G, S** (for Alt, Control, Shift, and Grey (numeric pad)). If modifier is followed by a **+** (plus), the key specification will be matched only if the modifier key is pressed. If the modifier is followed by **-** (minus), the state of the modifier key is ignored.

Keybinding examples

Here are some examples of key specifications (see configuration files under config/kbd for more):

[A]
Uppercase a

[a]
Lowercase a

[;]
Semicolon

[A+A]
Alt+A

[C+B]
Ctrl+B

[A+C+F1]
Alt+Ctrl+F1

[A+C+S+F1]
Alt+Ctrl+Shift+F1

[A+Space]
Alt+Space

[C+K_C+B]
Ctrl+K and then Ctrl+B (two keys)

[C+A_C+B_C+C]
Ctrl+A, Ctrl+B and Ctrl+C must be pressed in sequence.

[G+-]
Gray -

[G++]
Gray +

[C-S-X]
X, ignore the state of Ctrl and Shift keys.

[C+\\]
Ctrl+Backslash

[C+\[]
Ctrl+[

[C+G-Left]
Ctrl+Left, ignore difference between the two Left keys.

[C+A-A]
Ctrl+A, ignore the state of Alt key.

Abbreviations

Abbreviations are used to automatically replace some text or run an editor macro when some word is typed in. When a non-word character is entered, the previous word is searched for in the list of abbreviations. If it is found, the word is either replaced with a new string or a macro is executed.

Some examples of abbreviations:

```
abbrev 'wcsw' 'WinCreateStdWindow';

abbrev 'ifx' {
    KillWordPrev; InsertString 'if () {'; LineIndent; LineNew;
    InsertString '}'; LineIndent;
    MoveUp; MoveLineEnd; 3:MoveLeft;
    Fail; # do not insert typed character
}
```

The first one defines a replacement string, while the second one defines an editor macro to be run.

N.B.: For abbreviations to work, setting Abbreviations must be set to 1 for active mode.

Configuration reference

Sections, directives, and comments

The syntax for eFTE2 configuration sections is:

```
type [name[:parent]] { settings }
```

type

One of `object, color_palette, global, sub, colorize, mode, eventmap`.

name

Section `object` may have the privileged name `GLOBAL`, `color_palette` can't have a name, `sub` can't have a parent.

settings

Each section type has its own settings syntax.

Comments start with # anywhere in a line and last until the end of the line.

Note: comments are not allowed in lines starting with compiler directives `%if`, `%endif`, `%define`, or `%undefine`.

Sections and %if/%endif blocks can be enclosed in one another.

When split across several files, the configuration can incude other files via the `include` command. Its twin command `oinclude` is meant for optional files in which users should make their changes, so updating the base files leaves them intact. Including non-existing files will not halt loading the configuration if done via `oinclude`.

Syntax:

```
[o]include '[folder/]file.fte';
```

Strings

Strings can be specified using any of `'` `"` `/` characters.

Single quoted strings perform no substitution. To include `'` or \ in a string, it must be preceded with a backslash.

Double quoted strings perform the following substitutions:

- \t -> ^I, tab character
- \r -> ^M, CR
- \n -> ^J, LF
- \e -> ^[, escape character
- \v -> ^L, vertical tab
- \b -> ^H, backspace
- \a -> ^G, bell

Strings started by `/` character require no escaping (except for '/'). Mostly useful for specifying regular expressions without double backslashes that are necessary in single and double quoted strings. In turn,

regular expression operators must be escaped to be interpreted as literal characters while operating with regular expressions.

Regular expressions

This is the reference for the particular syntax and operators of regular expressions in eFTE2, which may be different from other implementations you are used to. In particular, eFTE2 currently lacks operators to indicate match lengths other than conditional 0 or 1, but there may be more differences, so please read carefully.

Also, regular expression match and replace can be conditioned by some flags. These flags are unrelated to regular expressions themselves, and are the same used for literal text in search and replace commands.

Match Operators

****

(Un)quotes the following character: alphanumeric characters gain a special meaning as described below, and non-alphanumeric operators are interpreted literally (f.e. "\." comes to mean "a single dot" instead of "any character").

\n

Matches a 0x0A (LF) character.

\r

Matches a 0x0D (CR) character.

\t

Matches a 0x09 (TAB) character.

\e

Matches an escape character (0x1B).

\s

Matches whitespace (CR, LF, TAB, SPACE) characters.

\S

Matches non-whitespace (the reverse of \s).

\w

Matches word character [a-zA-Z0-9].

\W

Matches non-word character.

\d

Matches a digit [0-9].

\D

Matches a non-digit.

\U
Matches uppercase characters (A-Z).

\L
Matches lowercase characters (a-z).

\x##
Matches specified hex value (\x0A, \x0D, \x09, etc.).

\o###
Matches specified octal value (\o000, \o015, etc.).

\N###
Matches specified decimal value (\N000, \N013, \N009, etc.).

\C
Starts case sensitive matching.

\c
Starts case insensitive matching.

^
Match a beginning of line.

$
Match an end of line.

.
Match any character.

<
Match beginning of word (word consists of [A-Za-z0-9]).

>
Match end of word.

[ ]
Specifies a class of characters ([abc123], [\]\x10], etc), any of which will be matched individually.

[-]
Specifies a range of characters ([0-9a-zA-Z_], [0-9], etc).

[^]
Specifies complement class ([^a-z], [^\-], etc).

?
Matches preceding pattern optionally (a?bc, filename\.?, $?, etc.).

|

Matches preceding or next pattern (a|b, c|d, abc|d). Only one character will be used as pattern unless grouped together using parenthesis (), or curly braces {}.

*****

Match zero or more occurances of preceding pattern. Matching is greedy and will match as much as possible.

+

Match one or more occurances of preceding pattern. Match is greedy.

@

Match zero or more occurances of preceding pattern. Matching is non-greedy and will match as little as possible without causing the rest of the pattern match to fail.

#

Match one or more occurances of preceding pattern. Matching is non-greedy.

{ }

Group patterns together to form complex patterns (f.e.: {abc}, {abc}|{cde}, {abc}?, {word}?).

( )

Group patterns together to form complex patterns. Also used to save the matched substring into the register which can be used for substitution operation. Up to 9 registers can be used.

Replacement Operators

****

Causes next character to alternate between being interpreted literally, or as a special character. F.e.: \0, or \\ to get a literal "\".

\0

Recalls entire matched pattern.

\1 to \9

Recalls stored substrings from registers (\1, \2, \3, to \9).

\n

Inserts a 0x0A (LF) character.

\r

Inserts a 0x0D (CR) character.

\t

Inserts a 0x09 (TAB) character.

\u

Convert next character to uppercase.

\l
: Convert next character to lowercase.

\U
: Convert to uppercase till \E or \e.

\L
: Convert to lowercase till \E or \e.

Global Settings

Note: Many of these have been lifted from the original FTE documentation without further verification. They are styled like this text.

The following settings can be used in the `object` **GLOBAL** section of the configuration. Some of the options are platform specific (to be fixed).

DefaultModeName
: Default mode name for loading/editing files. If not set or invalid, first mode in the configuration file will be used instead. By default set to 'PLAIN'.

CompletionFilter
: A regular expression. Files matching it are ignored when doing filename completion.

CompileRx
: Defines regular expressions and their subpattern indices to match when searching for errors and warnings in compilation output. First number is an index of the subpattern that matches filename. The second must match the line number, the third parameter is the regular expression to match to each line of the compiler output.

OpenAfterClose
: If set to 1, editor will prompt for another file when all files are closed.

SysClipboard
: When set to 1, editor will use external (PM, X11) clipboard instead of internal one.

ScreenSizeX
: Number of columns visible on screen or window.

ScreenSizeY
: Number of lines visible on screen or window.

CursorInsertStart
: Starting percentage of cursor size (from top) when in insert mode.

CursorInsertEnd
: Ending percentage of cursor size when in insert mode.

CursorOverStart

Starting percentage of cursor size when in overstrike mode.

CursorOverEnd

Ending percentage of cursor size when in overstrike mode.

SelectPathname

If set to 1, pathname will be selected by default when prompting for a file in FileOpen function. If set to 0, pathname will not be selected, this allows you to quickly type a new filename, without erasing an entire entryfield.

ShowMenuBar

If set to 1, main menu bar will be visible.

ShowVScroll

If set to 1, scroll bar will be visible.

ShowHScroll

If set to 1, scroll bar will be visible.

KeepHistory

If set to 1, last file position and input prompt history will be loaded on startup and saved on exit. Can be overriden with command line option '-h'.

LoadDesktopOnEntry

If set to 1, all files listed in desktop file in current directory or program directory will be loaded into eFTE2. The desktop file can be overriden with command line option '-d'.

If set to 2, desktop is only loaded (and saved) if there are no files specified on the command line.

SaveDesktopOnExit

If set to 1, desktop will be automatically saved when ExitEditor command is issued.

KeepMessages

If set to 1, compiler messages will be kept until deleted by user.

ScrollBorderX

Horizontal offset to the border before window starts scrolling.

ScrollBorderY

Vertical offset to the border before window starts scrolling.

ScrollJumpX

Scroll window by this many columns when cursor reaches scrolling border.

ScrollJumpY

Scroll window by this many lines when cursor reaches scrolling border.

C_*

Define the C mode smart indentation parameters.

See section on configuring C mode indentation.

REXX_Indent

Defines the REXX basic indentation level.

C mode Smart Indentation settings

C_Indent

Basic C indentation level.

C_BraceOfs

Brace '{' offset.

C_CaseOfs

Offset of case and default statements.

C_CaseDelta

Offsets of statements following case/default.

C_ClassOfs

Offset of public, private and protected.

C_ClassDelta

Offset of statements following public, private, protected.

C_ColonOfs

Offset of labels.

C_CommentOfs

Offset of comments.

C_CommentDelta

Offset of second line of comments.

C_FirstLevelWidth

Width of the first indentation level (indent of '{' in the function start).

C_FirstLevelIndent

Indentation of statements in the first indentation level.

C_ParenDelta

When ≥ 0, offset of continued text after '('. When set to -1, the offset is equal to position of '(' plus one.

Example 1:

```
class line {
public:                     // C_ClassOfs = 0
    line();                 // C_ClassDelta = 4
    ~line();
};
int main() {
    int x = 1;
    /*                      // C_CommentOfs = 0
     * check value          // C_CommentDelta = 1
     */
    puts("main");           // C_Indent = 4
    if (x)
    {                       // C_BraceOfs = 0
        switch (x) {
        case 1:             // C_CaseOfs = 0
            puts("ok");     // C_CaseDelta = 4
            break;
        }
    }
end:
    return 0;
}
```

Example 2:

```
class line {
  public:                       // C_ClassOfs = 2
    line();                     // C_ClassDelta = 2
    ~line();
};
int main() {
    int x = 1;
      /*                        // C_CommentOfs = 2
      ** check value            // C_CommentDelta = 0
      */
    puts("main");               // C_Indent = 4
    if (x)
    {                           // C_BraceOfs = 0
        switch (x) {
            case 1:             // C_CaseOfs = 4
                puts("ok");     // C_CaseDelta = 4
                break;
        }
    }
end:
```

```
    return 0;
}
```

Interface colors

The syntax for color settings is:

```
color_palette { { 'name', 'value' } [, …] }
```

name
Any symbolic name defined here for later use.

value
Either a colon-separated pair of symbolic names, or a space-separated pair of numeric values; the first item in 'value' is for background, the second one for foreground.

Numeric values of colors:

0 (High bit 0, RGB mask 000)
Black

1 (High bit 0, RGB mask 001)
Dark Blue

2 (High bit 0, RGB mask 010)
Dark Green

3 (High bit 0, RGB mask 011)
Dark Cyan

4 (High bit 0, RGB mask 100)
Dark Red

5 (High bit 0, RGB mask 101)
Dark Magenta

6 (High bit 0, RGB mask 110)
Orange

7 (High bit 0, RGB mask 111)
Pale Gray

8 (High bit 1, RGB mask 000)
Dark Gray

9 (High bit 1, RGB mask 001)
Blue

A (High bit 1, RGB mask 010)
Green

B (High bit 1, RGB mask 011)
Cyan

C (High bit 1, RGB mask 100)
Red

D (High bit 1, RGB mask 101)
Magenta

E (High bit 1, RGB mask 110)
Yellow

F (High bit 1, RGB mask 111)
White

Syntax highlighting

Syntax:
`colorize new[:old]` `{ settings }`
If specified, colorizer `<new>` inherits settings from `<old>`.
Settings:

`SyntaxParser = "<parser>";`

Activates the specified syntax parser for highlighting mode. `<Parser>` can be any of:

PLAIN
No syntax parser, only keyword highlighting is available.

SIMPLE
User configurable syntax parser will be defined.

C
REXX
HTML
PERL
MAKE
DIFF
For viewing output of **diff**.

MERGE
For editing output of **rcsmerge** (RCS, CVS).

IPF
Ada
MSG
For editing E-Mail.

SH
PASCAL
TEX
FTE
CATBS

For **VIEWING ONLY** of **nroff** formatted man-pages (formatted with backspaces).

`color { { color match } [, { color match }… ] };`

Each "color match" couples color names internal to the colorizer and symbolic color names defined globally:

`{ "highlight_color", "global_name" }`

`highlight_color` must be one of "Normal", "Keyword", "String", "Comment", "CPreprocessor", "Regexp", "Header", "Quotes", "Number", "HexNumber", "OctalNumber", "FloatNumber", "Function", "Command", "Tag", "Punctuation", "New", "Old", "Changed", "Control", "Separator", "Variable", "Symbol", "Directive", "Label", "Special", "QuoteDelim", "RegexpDelim".

`keyword "color_name" { "some_keyword" [, "other_keyword"[, …]]};`

Multiple keyword sets with different colors can be defined. `color_name` is always a global editor color name.

Configurable Syntax Parser

When **SyntaxParser** is set to "SIMPLE", the following commands can be used to configure the state machine used for parsing the text:

`h_state state_number { "color_name" }`

Defines a new state for the state machine.

States must be numbered sequentially from 0 without skipping any numbers.

The color denoted by internal name will be used for characters that are not matched by any transition string or keyword.

`h_wtype { next_state_if_matched, next_state_if_not_matched, next_state_if_otheronly, "state_flags", "keyword_charset" }`

Specifies the keyword matching parameters for current state. There can be only one **h_wtype** keyword per state.

h_wtype takes the following arguments:

`next_state_if_matched`

-1 to keep current state, or the number of the next state to go into if a keyword is matched.

`next_state_if_not_matched`

-1 to keep current state, or the number of the next state to go into if a string other than a keyword is found.

next_state_if_otheronly
-1 to keep current state, or the number of the next state to go into if only characters not in the specified charset are found.

state_flags

String containing zero or more of the following characters:

i
Keyword matching is performed case-insensitively.

keyword_charset
A character set such as 'a-zA-Z0-9_$@'.

h_trans { next_state, trans_flags, trans_match, 'color_name' }

`h_trans` defines a new state transition for current state. It takes the following parameters:

next_state
The number of next state to go if a match is successful.

trans_flags

Determines options for matching `trans_match`. Can contain zero or more of the following characters:

^
Matches only at beginning of line.

$
Matches only at end of line.

i
Match is case-insensitive.

s
`trans_match` is a character set. Matches only if the next character is part of the set. "-" is allowed in `trans_match` to specify ranges of characters, f.e: 'A-J0-9'. If "-" is part of the set, it must be the first character in it or escaped.

S
Same as 's' but next character must not be part of the set.

x
`trans_match` is a regular expression. If some part of expression is enclosed in parentheses, pointer is advanced up to the start of parenthesized match (WARNING: this can cause infinite loops).

-

After successfull match, the pointer is not advanced, matching will resume at the same position in next state. (WARNING: this can cause infinite loops).

<

The matched character(s) are tagged with current state number. This is important for proper operation of **MatchBracket** command. MatchBracket will only match braces tagged with same state number.

>

The matched character(s) are tagged with next state number.

q

On successful match quote the next character (the next character is not used for matching).

Q

On successful match quote the end of line (the end of line is not used for matching).

`trans_match`

When **S** or **s** options are used, a set of characters, any of which should be matched. When **S** or **s** options are not used, a string to be matched. If `x` is specified, `trans_match` is a regular expression with its corresponding matching operators and escaped characters.

`h_words "color name" { "some_keyword" [,"another_keyword" … }`

Specifies the set of keywords to match in this state. All characters in keywords must be part of the `keyword_charset` in **h_wtype** command for this mode.

Works the same way as `keyword` but keywords are for current state only.

Multiple keyword sets with different colors can be defined.

'-' can be used for color specifier to use the default keyword color specified in global settings.

Editing macros

A macro can be declared to have a name assigned for invocation, or be directly assigned to a menu or keybinding without being declared.

Syntax:

```
sub <macro name> { macro }
```

or

```
menu <menu name> {
    […;] item "<title>" { macro } [;…]
}
```

or

```
eventmap <mode…> {
    [...;] key <keybinding> { macro } [;...]
}
```

macro breakdown:

```
[n:][?]command[; more commands]
```

n
How many times command will be executed.

?
Flag: if present, continue macro execution even if command fails.

command
Any other macro previously declared, or editor internal command (see next section).

Internal commands

Note: Most of these have been lifted from the original FTE documentation without further verification. They are styled like this text.

Cursor movement

MoveDown
Move cursor to next line.

MoveUp
Move cursor to previous line.

MoveLeft
Move cursor to previous column.

MoveRight
Move cursor to next column.

MovePrev
Move cursor to previous character. Moves to end of the previous line if cursor is at the beginning of line.

MoveNext
Move cursor to next character. Moves to the beginning of next line if cursor is at the end of line.

MoveWordLeft
Move cursor to the beginning of the word on the left.

MoveWordRight
Move cursor to the beginning of the word on the right.

MoveWordPrev
Move cursor to the beginning of the previous word.

MoveWordNext
Move cursor to the beginning of the next word.

MoveWordEndLeft
Move cursor to the end of the previous word.

MoveWordEndRight
Move cursor to the end of the word on the right.

MoveWordEndPrev
Move cursor to the end of the previous word.

MoveWordEndNext
Move cursor to the end of the next word.

MoveWordOrCapLeft
Move cursor to the beginning of the word or capital letter on the right.

MoveWordOrCapRight
Move cursor to the beginning of the word or capital letter on the left.

MoveWordOrCapPrev
Move cursor to the beginning of the previous word or to previous capital letter.

MoveWordOrCapNext
Move cursor to the beginning of the next word or to next capital letter.

MoveWordOrCapEndLeft
Move cursor to the end of the word or capitals on the left.

MoveWordOrCapEndRight
Move cursor to the end of the word or capitals on the right.

MoveWordOrCapEndPrev
Move cursor to the end of the previous word or capitals.

MoveWordOrCapEndNext
Move cursor to the end of the next word or capitals.

MoveLineStart
Move cursor to the beginning of line.

MoveLineEnd
Move cursor to the end of line.

MovePageStart
Move cursor to the first line on current page.

MovePageEnd
Move cursor to the last line on currently page.

MovePageUp
Display previous page.

MovePageDown
Display next page.

MoveFileStart
Move cursor to the beginning of file.

MoveFileEnd
Move cursor to the end of file.

MovePageLeft
Scroll horizontally to display page on the left.

MovePageRight
Scroll horizontally to display page on the right.

MoveBlockStart
Move cursor to the beginning of selected block.

MoveBlockEnd
Move cursor to end beginning of selected block.

MoveFirstNonWhite
Move cursor to the first non-blank character on line.

MoveLastNonWhite
Move cursor to the last non-blank character on line.

MovePrevEqualIndent
Move cursor to the previous line with equal indentation.

MoveNextEqualIndent
Move cursor to the next line with equal indentation.

MovePrevTab
Move cursor to the previous tab position.

MoveNextTab
Move cursor to the next tab position.

MoveTabStart
When cursor is on the tab characters, moves it to the beginning of the tab.

MoveTabEnd
When cursor is on the tab characters, moves it to the end of the tab.

MoveLineTop
Scroll the file to make the current line appear on the top of the window.

MoveLineCenter
Scroll the file to make the current line appear on the center of the window.

MoveLineBottom
Scroll the file to make the current line appear on the bottom of the window.

ScrollLeft
Scroll screen left.

ScrollRight
Scroll screen right.

ScrollDown
Scroll screen down.

ScrollUp
Scroll screen up.

MoveFoldTop
Move to the beginning of current fold.

MoveFoldPrev
Move to the beginning of previous fold.

MoveFoldNext
Move to the beginning of next fold.

MoveBeginOrNonWhite
Move to beginning of line, or to first non blank character.

MoveBeginLinePageFile
Move to the beginning of line. If there already, move to the beginning page. If there already, move to the beginning of file.

MoveEndLinePageFile
Move to the end of line. If there already, move to the end page. If there already, move to the end of file.

MoveToLine
Move to line number given as argument.

MoveToColumn
Move to column given as argument.

MoveSavedPosCol
Move to column from saved position.

MoveSavedPosRow
Move to line from saved position.

MoveSavedPos
Move to saved position.

SavePos
Save current cursor position.

MovePrevPos
Move to last cursor position.

See also: all commands.

Deleting text

KillLine
Delete current line. If the line is the last line in the file, only the text is deleted.

KillChar
Delete character under (after) cursor.

KillCharPrev
Delete character before cursor.

KillWord
Delete the word after cursor.

KillWordPrev
Delete the word before cursor.

KillWordOrCap
Delete word or capitals after cursor.

KillWordOrCapPrev
Delete word or capitals before cursor.

KillToLineStart
Delete characters to the beginning of line.

KillToLineEnd
Delete characters to the end of line.

KillBlock
Delete block.

KillBlockOrChar
If block is marked, delete it, otherwise delete character under cursor.

KillBlockOrCharPrev
If block is marked, delete it, otherwise delete character before cursor.

Delete
Delete character under (after) cursor.

BackSpace
Delete character before cursor.

See also: DeleteKillTab, DeleteKillBlock, BackSpKillTab, BackSpKillBlock, all commands.

Line commands

LineInsert
Insert a new line before the current one.

LineAdd
Add a new line after the current one.

LineSplit
Insert line break at cursor position.

LineJoin
Supress break between current line and the next one. If cursor is positioned beyond the end of line, the current line is first padded with whitespace.

LineNew
Insert line break and move to the beginning of the new line.

LineIndent
Reindent current line.

LineTrim
Trim whitespace at the end of current line.

LineDuplicate
Duplicate the current line.

LineCenter
Center the current line.

See also: all commands.

Block commands

BlockBegin
Set block beginning to current position.

BlockEnd
Set block end to current position.

BlockUnmark
Unmark block.

BlockCut
Cut selected block to clipboard.

BlockCopy
Copy selected block to clipboard.

BlockCutAppend
Cut selected block and append it to clipboard.

BlockCopyAppend
Append selected block to clipboard.

BlockClear
Clear selected block.

BlockPaste
Paste clipboard to current position.

BlockKill
Delete selected text.

BlockIndent
Indent block by 1 character.

BlockUnindent
Unindent block by 1 character.

BlockMarkStream
Start/stop marking stream block.

BlockMarkLine
Start/stop marking line block.

BlockMarkColumn
Start/stop marking column block.

BlockExtendBegin
Start extending selected block following cursor moves.

BlockExtendEnd
Stop extending selected block.

BlockReIndent
Reindent entire block (C/REXX mode).

BlockSelectWord
Select word under cursor as block.

BlockSelectLine
Select current line as block.

BlockSelectPara
Select current paragraph (delimited by blank lines) as block.

BlockPasteStream
Paste clipboard to current position as stream block.

BlockPasteLine
Paste clipboard to current position as line block.

BlockPasteColumn
Paste clipboard to current position as column block.

BlockPrint

Print a block to configured device.

BlockRead

Read block from file.

BlockReadStream

Read block from file as stream block.

BlockReadLine

Read block from file as line block.

BlockReadColumn

Read block from file as column block.

BlockWrite

Write marked block to file.

BlockSort

Sorts the marked block in ascending order. If mode setting MatchCase is set, characters will be compared case sensitively. When block is marked in Stream or Line mode, the entire lines in marked block will be compared. When block is marked in Column mode, only characters within marked columns will be compared.

BlockSortReverse

Sorts the marked block in descending order.

BlockUnTab

Remove tabs from marked lines.

BlockEnTab

Generate and optimize tabs in marked lines.

BlockMarkFunction

Mark current function as block.

BlockTrim

Trim end-of-line whitespace.

See also: all commands.

Text editing

Undo

Undo last operation.

Redo
: Redo last undone operation.

See also: all commands.

Folding text

FoldCreate
: Create fold.

FoldCreateByRegexp
: Create folds at lines matching a regular expression.

FoldCreateAtRoutines
: Create folds at lines matching RoutineRegexp.

FoldDestroy
: Destroy fold at current line.

FoldDestroyAll
: Destroy all folds in the file.

FoldPromote
: Promote fold to outer level.

FoldDemote
: Demote fold to inner level.

FoldOpen
: Open fold at current line.

FoldOpenNested
: Open fold and nested folds.

FoldClose
: Close current fold.

FoldOpenAll
: Open all folds in the file.

FoldCloseAll
: Close all folds in the file.

FoldToggleOpenClose
: Toggle open/close current fold.

See also: all commands.

Bookmarks

PlaceBookmark
Place a file-local bookmark.

RemoveBookmark
Remove a file-local bookmark.

GotoBookmark
Go to file-local bookmark location.

PlaceGlobalBookmark
Place global (persistent) bookmark.

RemoveGlobalBookmark
Remove global bookmark.

GotoGlobalBookmark
Go to global bookmark location.

PushGlobalBookmark
Push global bookmark (named as #<num>) to stack.

PopGlobalBookmark
Pop global bookmark from stack.

See also: all commands.

Character translation / insertion

CharCaseUp
Convert current character to uppercase.

CharCaseDown
Convert current character to lowercase.

CharCaseToggle
Toggle case of current character.

CharTrans
Translate current character (like perl/sed).

LineCaseUp
Convert current line to uppercase.

LineCaseDown
Convert current line to lowercase.

LineCaseToggle
Toggle case of current line.

LineTrans
Translate characters on current line.

BlockCaseUp
Convert characters in selected block to uppercase.

BlockCaseDown
Convert characters in selected block to lowercase.

BlockCaseToggle
Toggle case of characters in selected block.

BlockTrans
Translate characters in selected block.

InsertString <string>
Insert argument `string` at cursor position.

InsertSpace
Insert space.

InsertChar
Insert character argument at cursor position.

TypeChar
Insert character at cursor position (expanding abbreviations).

InsertTab
Insert tab character at cursor position.

InsertSpacesToTab
Insert appropriate number of spaces to simulate a tab.

SelfInsert
Insert typed character.

WrapPara
Wrap current paragraph.

InsPrevLineChar
Insert character in previous line above cursor.

InsPrevLineToEol
Insert previous line from cursor to end of line.

CompleteWord
Complete current word to last word starting with the same prefix.

See also: all commands.

File commands

FileOpen
Prompts for file name to open, editing mode is determined automatically. If a directory name is specified, the internal file browser is opened.

FileOpenInMode "<mode>"
Prompts for file name to open using specified mode.

FileReload
Reload current file.

FileSave
Save current file.

FileSaveAll
Save all modified files.

FileSaveAs
Rename Save current file.

FileWriteTo
Write current file into another file.

FilePrint
Print current file.

FileClose
Close current file.

FileCloseAll
Close all open files.

FileTrim
Trim end-of-line whitespace.

FilePrev
Switch to previous file in ring.

FileNext
Switch to next file in ring.

FileLast

Exchange last two files in ring.

SwitchTo

Switch to numbered buffer given as argument.

See also: all commands.

Directory commands

DirOpen

Open directory browser.

DirGoUp

Change to parent directory.

DirGoDown

Change to currently selected directory.

DirGoRoot

Change to root directory.

DirGoto

Change to directory given as argument.

DirSearchCancel

Cancel search.

DirSearchNext

Find next matching file.

DirSearchPrev

Find previous matching file.

RenameFile

MakeDirectory

See also: all commands.

Search and replace

IncrementalSearch

Incremental search.

`Find [find [flags]]`
`FindReplace [find [replace [flags]]]`

`Find` and `FindReplace` will interact with the user when parameters are omitted. The editor then asks what to find, [what to replace it with,] and for the search flags string: "Options (All / Block / Cur / Delln / Glob / Igncase / Joinln / Rev / SplitLn / Noask / Word / regX):".

Notes:

1. `find` and `replace` can be regular expressions.
2. In configuration files, `find`, `replace`, and `flags` are either single- or double-quote-, or slash-delimited character strings in which the usual replacements take place.

`flags` string may contain the following characters:

a
"All matches." If absent only one Find/Replace operation will be performed. If present, a Find operation will move cursor position to the last match.

b
Operate within the selected block only (cursor must be located ahead of block start).

c
Include matches at cursor position; default is to skip to next match if any.

d
Delete line containing matched find -> same as replacing with "" + LineDelete would do.

g
Global lookup, i.e. from beginning of file.

i
Ignore case when matching.

j
Join line containing matched find with the next line.

r
Reverse look up (upstream in the file, only for literal Find/Replace).

s
Split line containing match at cursor position.

n
Do not ask for confirmation (this tag is honored only when invoking Find/Replace commands from a configuration file, and not when taken from user input?).

---

x
Regular expression; if not present, find [and replace] are interpreted as literal strings.

w
"Words only" ?

FindRepeat
Repeat last find/replace operation, including flags.

FindRepeatOnce
Repeat last find/replace operation only once.

FindRepeatReverse
Repeat last find/replace operation in reverse. Not supported in RegExp mode (yet?), only literal search works.

MatchBracket
Find matching bracket ([{<>}]).

HilitWord
Highlight current word everywhere in the file.

SearchWordPrev
Search for previous occurence of word under cursor.

SearchWordNext
Search for next occurence of word under cursor.

HilitMatchBracket
Highlight matching bracket.

See also: all commands.

Window commands

WinHSplit
Split window horizontally.

WinNext
Switch to next (bottom) window.

WinPrev
Switcn to previous (top) window.

WinClose
Close current window.

WinZoom
Delete all windows except for current one.

WinResize

Resize current window (+n,-n given as argument).

ViewBuffers

View currently open buffers.

ListRoutines

Display routines in current source file.

ExitEditor

Exit eFTE2.

ShowEntryScreen

View external program output if available.

See also: all commands.

Compiler support

Compile [command]

Ask for compile command and run compiler. If `command` parameter is specified, it is offered to the user for completion.

RunCompiler

Run configured compile command.

ViewMessages

View compiler output.

CompileNextError

Switch to next compiler error.

CompilePrevError

Switch to previous compiler error.

RunProgram

Run external program.

See also: all commands.

CVS support

Cvs

Ask for CVS options and run CVS.

RunCvs
: Run configured CVS command.

ViewCvs
: View CVS output.

ClearCvsMessages
: Clear CVS messages.

CvsDiff
: Ask for CVS diff options and run CVS.

RunCvsDiff
: Run configured CVS diff command.

ViewCvsDiff
: View CVS diff output.

CvsCommit
: Ask for CVS commit options and run CVS.

RunCvsCommit
: Run configured CVS commit command.

ViewCvsLog
: View CVS log.

See also: all commands.

TAGS commands

eFTE2 supports TAGS files generated by programs like ctags.

TagFind
: Find word argument in tag files.

TagFindWord
: Find word under cursor in tag files.

TagNext
: Switch to next occurance of tag.

TagPrev
: Switch to previous occurance of tag.

TagPop
: Pop saved position from tag stack.

TagLoad

Load tag file and merge with current tags.

TagClear

Clear loaded tags.

TagGoto

See also: all commands.

Option commands

ToggleAutoIndent
ToggleInsert
ToggleExpandTabs
ToggleShowTabs
ToggleUndo
ToggleReadOnly
ToggleKeepBackups
ToggleMakeBackups
ToggleMatchCase
ToggleBackSpKillTab
ToggleDeleteKillTab
ToggleSpaceTabs
ToggleIndentWithTabs
ToggleBackSpUnindents
ToggleWordWrap
ToggleTrim
ToggleShowMarkers
ToggleHilitTags
ToggleShowBookmarks
SetLeftMargin
SetRightMargin
ToggleSysClipboard
SetPrintDevice
ChangeTabSize
ChangeLeftMargin
ChangeRightMargin

See also: all commands.

Miscellaneous commands

ShowPosition
: Show internal position information on status line.

ShowVersion
: Show editor version information.

ShowKey
: Wait for keypress and display modifiers+key pressed in status line.

WinRefresh
: Refresh display.

MainMenu
: Activate main menu.

ShowMenu
: Popup menu specified as argument.

LocalMenu
: Popup context menu.

ChangeMode
: Change active mode for current buffer.

ChangeKeys
: Change keybindings for current buffer.

ChangeFlags
: Change option flags for current buffer.

Cancel

Activate

Rescan

CloseActivate

ActivateInOtherWindow

DeleteFile

ASCIITable
: Display ASCII selector in status line.

DesktopSave
: Save desktop.

ClipClear
: Clear clipboard.

DesktopSaveAs
: Save desktop under a new name.

DesktopLoad

ChildClose

BufListFileSave
: Save currently selected file in buffer list.

BufListFileClose
: Close currently selected file in buffer list.

BufListSearchCancel
: Cancel search.

BufListSearchNext
: Next match in search.

BufListSearchPrev
: Previous match in search.

ViewModeMap
: Lists current mode keybindings in EventMapView.

ClearMessages
: Clear compiler messages.

IndentFunction
: Indent current function.

MoveFunctionPrev
: Move cursor to previous function.

MoveFunctionNext
: Move cursor to next function.

InsertDate
: Insert date at cursor.

InsertUid
: Insert user name at cursor.

FrameNew

FrameClose

FrameNext

FramePrev

BufferViewNext

BufferViewPrev

ShowHelpWord "FileList?" OS-specific?

Show context help on keyword.

ShowHelp [file? [what?]]

Show help manual.

ConfigRecompile

Recompile editor configuration.

SetCIndentStyle

Set C indentation style parameters. Has the following parameters: C_Indent = 4; C_BraceOfs = 0; C_ParenDelta = -1; C_CaseOfs = 0; C_CaseDelta = 4; C_ClassOfs = 0; C_ClassDelta = 4; C_ColonOfs = -4; C_CommentOfs = 0; C_CommentDelta = 1; C_FirstLevelWidth = -1; C_FirstLevelIndent = 4; C_Continuation = 4;

SetIndentWithTabs

Set value of indent-with-tabs to argument.

ListMark

Mark single line in list.

ListUnmark

Unmark single line in list.

ListToggleMark

Toggle marking of single line in list.

ListMarkAll

Mark all lines in list.

ListUnmarkAll

Unmark all lines in list Toggle marking of all lines in list.

See also: all commands.

Menus

Syntax:

```
  menu <name> { [item ["<title>" { macro }]]
[; [more items]
```

```
[submenu "<title>", <submenu>]
[; more items]]
}
```

Menu item definitions are separated by semicolons. This is optional for the last menu item. Items without parameters are inactive and split their menu into groups.

<name>
An internal name to refer to the menu from event maps or other menus.

<submenu>
The `name` of another menu to be opened from this menu item.

Modes

The syntax of **mode** definitions is:

```
mode NEW[:PARENT] { mode settings }
```

Mode named `NEW` inherits its settings from mode `PARENT` if specified at mode declaration.

Mode Settings

The following settings can be specified for each mode:

ExpandTabs {0,1}
Should be set to 1 if tabs are to be expanded when displayed. Use ToggleExpandTabs command to toggle during editing.

TabSize {1-32}
Tab size when tabs are shown expanded on display.

AutoIndent {0,1}
Should be set to 1 if autoindent is to be used. Use ToggleAutoIndent command to toggle it on/off during editing.

Insert {0,1}

If set to **1**, Insert mode is active by default. If set to **0**, overwrite mode is active by default.

Use ToggleInsert command to toggle it on/off during editing.

StripChar {ASCII code/-1}

This character will be stripped if found at the end of any lines when the file is being loaded. If it is set to **-1**, no characters will be stripped.

Normally used to strip 13 (CR) characters from DOS text files.

LineChar {ASCII code/-1}

This character is used as a line separator when loading a file. If set to **-1**, there is no line separator. (**WARNING: File will be loaded as one line if LineChar is set to -1**).

Usually set to 10 (LF) as standard text file line separator.

AddCR {0,1}
If set to **1**, CR (13, \r) character will be added to end of line when saving.

AddLF {0,1}
If set to **1**, LF (10, \n) characted will be added to end if line when saving.

ForceNewLine {0,1}
Normally, the last line is saved without any CR/LF characters when saving. This setting will override that behaviour.

Hilit {0,1}
If set to **1**, syntax highliting will be active.

ShowTabs {0,1}
If set to **1**, tabs will be visible as circles (EPM-like).

IndentMode {PLAIN,C,REXX}
Activates the specified smart indent mode. (PLAIN mode specifies no smart audoindenting, just normal autoindent).

Colorizer <name>
Specifies a previously declared colorize to use for syntax highlighting in current mode.

EventMap <name>
Specifies the existing `name` eventmap to use in current mode.

UndoLimit {Number}
Limit undo to this many recent commands (-1 = unlimited)

UndoMoves {0,1}
If set to 1, all cursor movements will be recorded on undo stack.

MakeBackups {0,1}
If set to **1**, backup files will be created. KeepBackups determines if they are kept after save is successful.

KeepBackups {0,1}
If set to **0**, backup files will be deleted after a successful save operation.

MatchCase {0,1}
If set to **0**, searches will be case insensitive. This can be toggled via ToggleMatchCase command when editor is running.

BackSpKillTab {0,1}
If set to 1, BackSpace will kill entire tabs instead of converting them to spaces.

DeleteKillTab {0,1}
If set to 1, Delete will kill entire tabs instead of converting them to spaces.

BackSpUnindents {0,1}
If set to 1, BackSpace will unindent to previous indentation level if issued on beginning of line.

SpaceTabs {0,1}
If set to 1, InsertTab command will insert spaces instead of tabs.

IndentWithTabs {0,1}
If set to 1, indentation will be done using tabs instead of spaces.

WordWrap {0,1,2}
If set to 1, editor wrap current line when right margin is reached. If set to 2, editor will wrap current paragraph continously. Paragraphs as recognised by WrapPara command must be separated by blank lines.

LeftMargin {1-xx}
Left margin for word wrap (WrapPara command).

RightMargin {1-xx}
Right margin for word wrap (WrapPara command).

Trim {0,1}
If set to 1, spaces on the end of line will be trimmed when editing.

ShowMarkers {0,1}
If set to 1, end of line and end of file markers will be shown.

CursorTroughTabs {0,1}
If set to 1, editor will allow cursor to be positioned inside tabs.

DefFindOpt 'options'
Default search flags for Find command.

DefFindReplaceOpt 'options'
Default search/replace flags for FindReplace command.

SaveFolds {0,1,2}
If 0, folds are not saved. If 1, folds are saved at beginning of line, if 2 folds are saved at end of line.

Folds are saved as comments in source files, depending on active editing mode.

See mode settings CommentStart and CommentEnd for configuration of fold comments.

CommentStart "comment-start-string"
String that starts comments (for saving folds)

CommentEnd "comment-end-string"
String that ends comments

AutoHilitParen {0,1}
If set to **1**, editor will automatically highlight the matching bracket if it is visible on screen. This is equivalent to executing the command HilitMatchBracket.

Abbreviations {0,1}
If set to 1, abbreviation expansion will be active in this mode.

BackSpKillBlock {0,1}
If set to 1, BackSpace command will delete block if it is marked, otherwise it will delete the previous character.

DeleteKillBlock {0,1}
If set to 1, Delete command will delete block if marked, instead of deleting the character below cursor.

PersistentBlocks {0,1}
If set to 1, blocks will stay marked even if cursor is moved in the file, if set to 0, block is unmarked as soon as the cursor is moved.

InsertKillBlock {0,1}
If set to 1, the marked block is deleted when a new character is typed.

FileNameRx "regexp"
Must be set to <regexp> matching names of files that should be edited in this mode. Has priority over FirstLineRx

FirstLineRx "regexp"
Must be set to <regexp> matching the first line of files that should be edited in this mode. This is checked only if no FileNameRx in any mode matches the filename.

RoutineRegexp "regexp"

Regular expression that matches function headers or otherwise relevant logical blocks of the file being edited.

Used by editor commands: ListRoutines, MoveFunctionPrev, MoveFunctionNext, BlockMarkFunction, IndentFunction.

Examples of appropriate settings for loading a number of file formats are given in the customization section of the manual.

Eventmap

`eventmap` sections are declared with the syntax:

```
eventmap mode[:parent] {
    [MainMenu = 'name'; ]
    [LocalMenu = 'name'; ]
    [keybindings]
    ...
```

```
    [abbreviations]
    …
}
```

Menu settings

`MainMenu 'name'`
Sets menu defined with `name` in the configuration as the main application pull-down menu displayed when working in this mode.

`LocalMenu 'name'`
Sets menu defined with `name` in the configuration as the local, pop-up menu used when working in this mode.

Keybindings

Keys are bound using the `key` command with the following syntax:

`key [ keyspec ]` { macro }

`keyspec` (non-optional, but delimited by "[" and "]") is a key name, or sequence of them: multiple-key combinations can be specified by separating them with "_" (underline).

Key names are denoted by the characters they return, or the following special key names (case sensitive): **F1-F12, Home, End, PgUp, PgDn, Insert, Delete, Up, Down, Left, Right, Enter, Esc, BackSp, Space, Tab, Center (meaning 5 in the numeric pad).**

ASCII characters ≥ 32 are bound to TypeChar by default.

Key names can be preceded by modifiers **A, C, G, S** plus "+" or "-" flag. **-** (minus) indicates to ignore modifier key, **+** (plus) indicates to match keyname only if previous modifier key is pressed.

A number of keybinding examples is given in the Customizing section of the manual.

Abbreviations

Syntax:

```
  abbrev 'old-word' 'new-string';
abbrev 'old-word' { macro }
```

N.B.: For abbreviations to work, the setting Abbreviations must be 1 for active mode.

This program, authors, timeline

WWW: http://svn.netlabs.org/efte

This is the first version of eFTE2. It is based on eFTE (see README.efte and http://sourceforge.net/projects/efte/) which in turn is based on FTE (Folding Text Editor, see http://fte.sourceforge.net/). The configuration now compiles on the fly.

License of use

eFTE2, Version 1.0

Copyright 2009–2016 by Gregg Young
Copyright 2008–2009 by eFTE SF Group.
Copyright 1994–1998 by Marko Macek.
All rights reserved.

This program is free software; you can redistribute it and/or modify it under the terms of either:

- the GNU General Public License as published by the Free Software Foundation; either version 2, or (at your option) any later version, or
- the "Artistic License" which comes with this Kit.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See either the GNU General Public License or the Artistic License for more details.

You should have received a copy of the Artistic License with this Kit, in the file named "Artistic". If not, I'll be glad to provide one.

You should also have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

GNU GENERAL PUBLIC LICENSE Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.
?

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices

stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

?

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three

years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.
?
4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the

Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
?

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

?

Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright © 19yy <name of author>
>
> This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
>
> This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

> Gnomovision version 69, Copyright © 19yy name of author
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
> This is free software, and you are welcome to redistribute it
> under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items—whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

```
  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice
```

```
This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Library General
Public License instead of this License.
```

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package"
refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version"
refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder"
is whoever is named in the copyright or copyrights for the package.

"You"
is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee"
is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available"
means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
    a. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
    b. use the modified Package only within your corporation or organization.
    c. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
    d. make other distribution arrangements with the Copyright Holder.
4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:
    a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.
    b. accompany the distribution with the machine-readable source of the Package with your modifications.
    c. give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.
    d. make other distribution arrangements with the Copyright Holder.
5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.
6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whomever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package via the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of

this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.
8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.
9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.
10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

Authors

Current Author

- Gregg Young

Current Contact Method:

- http://svn.netlabs.org/efte
- ygk@qwest.net

Earlier Authors

- Marko Macek
- Jeremy Cowgar
- Lauri Nurmi
- Timo Sirainen

Contributors

- F.Jalvingh
- Markus F.X.J. Oberhumer
- Martin Frydl
- S. Pinigin
- Zednek Kabelac
- Don Mahurin
- Lothar Schmidt
- Alfredo Fernández Díaz
- Michael DeBusk
- Timo Maier
- John Small

If you feel your name should be listed but it isn't, please contact us.

Future, present and past: revision history

Plans (by Gregg Young)

In the future you should be able to use eFTE2 with the configuration files of an existing FTE install again simply by creating a `mymain.fte` file in the same directory as your old `main.fte`.

I have structured the configuration files to make it easier to translate the menus. I lack the language skills to do the translations. If you have an opportunity to do one or more of the translations please send them to me at ygk@qwest.net. The menu files are located in the `menu_xx` (where xx is the language code) subdirectories of the config directory. You may add any language with a 2 letter code. Simple create a `menu_xx` subdirectory using the code and place the menus in it. To test them copy the direcctory `menu_xx` to the config directory of an eFTE2 install and start eFTE2 with -Lxx (xx = language code) on the command line. The `edefault.fte` also could be translated. If you are interested in translating the documentation (help file, etc.) please contact me.

Known issues

1. Occasionally, eFTE2 will trap when starting an external program. The problem appears to be an estyler issue. If this is a significant problem for you try adding eFTEPM.exe to estyler's exclusion list.
2. Opening some help files with certain keywords will trap newview. This is an issue either with newview or with the help file as it occurs when newview searches the same term. (e.g. open the Open Watcom 2.0 C Library Reference and search for return).
3. Anything placed on the command-line after --debug<clean> is ignored. If you wish to use other parameters and or load files with `--debug<clean>` place them before it.

Revision legend

-
: general alteration/note

+
: added new feature

!
: fixed a bug

C
: Configuration file Changes

As eFTE2

eFTE/2 1.0 — May 30, 2016

- First release of eFTE2 (C)
- Branded eFTE to eFTE/2
- Migrated to netlabs SVN
- Renamed some configuration files and reorganized configuration directories. (C)
- Program will still work with old configurations (C)
- Build with OpenWatcom
+ Changes to ease translation of the menus (C)
+ Added Enhanced HTML, CSS, and IPF modes by Alfredo Fernández Díaz (C)
+ Added Spanish translation by Alfredo Fernández Díaz (C)
+ Added Enhanced REXX editing mode by Michael DeBusk (C)
+ Added PHP & MySQL modes by Timo Maier (C)
+ Added BAT, BTM, and CONFIG.SYS modes by Michael DeBusk (C)
+ Added %undefine configuration compiler directive (C)
+ Reactivate the status bar message for toggles (e.g. tells you word wrap has changed to "Auto", "Yes" or "No")
+ Updated main.fte to include all available modes (C)
+ Moved UI %defines to mymain.fte with possibility to override them in systemmain.fte (C)
+ Moved color %defines to mymain.fte (C)
! Fixed error messages to always use a dialog box for gui versions
! Tolerate white space after "%" directives in configuration files (C)
+ Added ability to set the "C" indent style from command-line or by an environment variable
+ Restored a default configuration (edefault.fte) (C)
+ Move default font size settings to global.fte added myfontsize.fte to allow for over riding the default (C)
! Fixed the find dialog layout
+ Added latest perl highlighting code from FTE
+ Updated help file to reflect changes converted HTML to IPF
+ Help contents menu item using F1 accel key to open updated help file
+ Added wis file highlighting (C)
+ Warpin installer
! Improve ability of eFTE to find its config files
* Added environment variable to set config directory
+ Added REXX_End_Offset Effects placement of end, catch and finally relative to do, loop and select; 0 = aligned (C)
1 Fixed reverse search, search for all (trapped PM version) and search and replace where the search string was contained in the replace string
+ Pass search string to help files (either the word at the cursor or from a dialog if no word is picked)
! Clean up appearance of help viewer opening in PM
+ Make indent level for REXX keywords end, catch and finally user settable (C)
+ Added -I to default grep command (C)
+ Added rescan, compile, grep and run program button to the toolbar.
+ Added bubble help to toolbar buttons

* Added exceptq to the executables
+ Added exceptq TRP file support (C)
+ New Icons for program objects
! Update usage instruction provide a dialog to show them for PM version
+ Lxlite executables
+ Got the about dialog working in PM version updated its appearance
+ Add build level strings to exes
+ "Wrap" large menus in PM version removing "mode more" menus in the process
+ Added previous and next menu items
! Multiple changes to avoid/fix buffer overruns, hangs and traps
! Got -H and -D command-line switches to accept filenames
! Fixed scrollbar failure on files with more than 64k lines
- Added eFTE to PATH and eFTE help to HELP in config.sys
* Added OS/2 resource file highlighting (C)
* Added Help menu items for multiple file types including REXX, C, EMX, WIS, WMAKE and more (C)
! Fixed wrong month in log timestamps
* Added support for eCS/OS2 HOME (Steven Levine)

As eFTE

1.1—October 11, 2009

! Fixed a frequent buffer overflow in e_redraw.cpp
! Fixed a couple of memory/resource leaks found by static analysis

1.0—June 16, 2009

- First release of eFTE
- Branded FTE to be eFTE
- Removed DJGPP support
- Migrated to SVN for revision control
+ Added CursorWithinEOL
+ Added searching in the Routine List
+ Added new mode for CMake files
+ Windows XP and Vista now expand ~ to be the users home directory
+ Windows now normalizes path separators, / becomes \\
! BlockRead command was not expanding any file
! WM_COMMAND now set in X11 binary
- Removed partially implemented block cursor sizing
+ Added DesktopLoad command
+ Added oinclude command which optionally includes a configuration file
+ Modes are now user configurable w/o need to edit actual mode file
- Cleaned up code a bit with manual and automatic formatting (astyle)
- Converted from Makefiles to CMake on Win32, Unixes and OS/X
! assert bug fixed when compiling with newer Microsoft compilers
+ Configuration compiler loading is now much smarter

- Moved all documentation to the wiki where it is now user editable as well as developer editable.
! Many minor bug fixes
- Began native Mac OS X GUI interface
+ Added configuration variable BackupDirectory to allow storing backups in a common directory
+ Added based modes SOURCE and MARKUP and changed all existing modes to inherit from SOURCE, MARKUP or PLAIN
+ ebuild added, contributed by Daniel Hiepler
+ efte.spec added for rpm distributions
+ Macro system is now case insensitive
+ Added split method to FindReplace
+ Removed binary configuration dependency. eFTE now reads the plain text configuration files
+ Added a nicer help screen
! WM_HINTS are set properly in X11
+ GetString and $Str0-$Str9
+ User defined mode based generic indentation system
+ RegExp macro to perform regular expression replacements on user variables
+ ExpandTemplate macro for advanced user templates
+ Added user defined regular expression based indentation engine for any mode
+ Added new modes for Euphoria, Lua, VHDL
! Many bugs fixed in the bash syntax parser

0.50.1—January 17, 2008

- Forked from FTE (http://fte.sourceforge.net)

As FTE

0.50.1—September 2006

- Cleanup and Debian package update

0.50.0—<fill month here> 2003

- Many internal and some external changes
- Some bugs fixed in this release
+ cfte: added -p[reprocess] command line paramter, it can be used to debug configuration files.
! X11: XShellCommand is used to specify used shell under X11. $TERM is no longer used as shell.

0.46

! bug fixes
! coredump when $DISPLAY not set fixed
! occasional coredump at exit from PM version fixed.
! X11: check for invalid -geometry (larger than 255x255).
+ colors are specified using a palette in the configuration file

+ ShowHelp command (view .INF file under OS/2, .HLP under Win*, manpage under UNIX). Context Sensitive.
+ configuration file preprocessor %if(), %endif, %define
+ Global (persistent) Bookmarks commands + Push/Pop bookmark
+ Under UNIX it should now print using lpr
+ SIOD mode contributed.
+ command FileTrim,BlockTrim - trim whitespace at end of lines
+ mode option to strip whitespace at EOF on FileSave. (TrimOnSave)
+ C/C++ indentation style is now selectable from menu
+ compile command configurable per mode (CompileCommand option)
* PM toolbar compile tool configurable (CompileCommand2 option)
+ command to compile without asking anything (RunCompiler)
+ only load desktop when no arguments on command line (option) (this is achieved by setting LoadDesktopOnEntry=2 in `global.fte`)
+ create folds with RoutineRx (command FoldCreateAtRoutines)
+ command to center current line (LineCenter)
+ OS/2: does not need 'clipserv' anymore (experimental, please report bugs).
+ Readonly files are not editable when loaded.
+ …

0.45—February 1997

! bug fixes.
! some command line option changes (-h = help now, -H = history).
+ support for multiple frames in the PM version.
+ first win32 console version.

0.44—November 28

0.44b6—November 1996

! Bug fixes in Linux pipe handling.
! Bug fixes in OS/2 PM version.
! Fixed repainting bugs in Messages view.

0.44b5—November 1996

! Minor bug fixes.
! Changes in syntax highlighting configuration.

0.44b4—October 1996

! Minor bug fixes and numerous performance improvements.
+ New commands: BlockEnTab, BlockUnTab.
+ Configurable syntax highlighting. Modes HTML/IPF/Ada/Pascal/… are now configured externally.

+ CTags support. New commands: TagFind, TagFindWord, TagNext, TagPrev, TagPop, TagLoad, TagClear. Needs external ctags utility to create tags file. Tagfile and tag to find can be given on command line.
+ New option: KeepMessages. New command: ClearMessages.
+ X11: added support for selection copy/paste.
+ PM: Accept file dropped on editor window.
+ PM: Optional toolbar (not configurable yet). New option: ShowToolBar.
+ PM: GUI dialogs (find/replace, file, …). New option: GUIDialogs.
+ PM: Conditional cascade menus can now be used.
+ PM: Alt+Fx accelerators can now be disabled with PMDisableAccel option.
- Removed WSStyleSearch flag.
! PM: Rollup of editor window should now work (tested with title.dll)
! Menus can now be overriden by predefining them.
! Unix: completion of .* (dot) files now works.
! Various fixes to C-mode smart indentation.
! BlockReadXXX caused crashes when used with bad filename.
+ Incremental search can now be continued by using up/down arrow.
! ExitEditor doesn't close files immediatelly after discard. Desktop is now properly saved after cancelling ExitEditor command.
+ PM: Bigger file selection dialog box with history and save position.
+ New commands: IndentFunction, BlockMarkFunction, MoveFunctionPrev, MoveFunction-Next. Contributed by: jalving@ibm.net
+ …

0.43—15 July 1996

! Minor bug fixes.
! Upper/Lower block in column mode could cause a crash.
! CompleteWord command occasionally inserted garbage when previous match was found in the same line.

0.42—July 1996

! Several minor bug fixes.
! PM version doesn't crash when non-existant file is loaded on startup.
! HilitWord command now works again.
+ Setting for HilitWord color.
+ ViewModeMap command is back.

0.41—June 1996

! Compile command crashed depending on command input.
! Substrings were matched for keywords in smart indentation (C,REXX)
! InsertSpacesToTab command always returned fail status.
! CFTE now compiles to temporary file first and replaces original on success only.
! CFTE returns correct errorlevel on failure (0 = ok, 1 = fail).
! SavePos/PrevPos is now stored using real line number, not virtual (folded).

+ BlockSort command.
+ UndoMoves setting can be set for mode to enable undo/redo of all cursor movements.
+ BlockCutAppend, BlockCopyAppend commands to append cut/copied block to clipboard.
+ Error message locations now track the position better when a file is edited (lines are added/removed).

0.40—June 1996

! Keyword inheritance was not properly handled in colorize modes.
! Colors were not inherited in colorize modes.
! OS2: Keys Alt+<menu-letter>, F10, Alt+Enter, Alt+Space are available for remapping.
! Some startup window sizing problems fixed.
! Fixed several bugs in configuration files (no bindings for MSG mode, some Alt+<letter> menu shortcuts).
! Abbreviation expansion could abort with 'asssertion failed'.
+ Performance improvements (MatchBracket and related stuff).
+ OS2: Window position is now saved.
+ FTE now remembers the directory the compilation was started from and will resolve all relative pathnames found in error messages using this directory. The current directory is determined by currently active file or directory. If Messages are already open, FTE will use directory from there instead of the current one. The current directory of message list is always the directory the compilation was started from.

0.39—May 96

! Fixed crash in Compile commands when repeating it.
! Multi-key bindings inserted an ascii char if the 2+ key was not valid.
! when inserting) in Cmode, it failed to advance the cursor when there was no match and AutoHilitParen was set to 1.
+ EventMap variable to define keymap to use for mode.
! MoveToLine shows correct default line value when folds are used.
- Did some reorganization of config files to make adding new binding sets easier.

0.38—May 96

! Many bug fixes (mostly minor).
+ BlockWrite command can now append to a file.
+ Directory browser.
! PERL: properly highlight `s[][]`, `tr[][]`.
+ Configuration files must now be compiled.
+ Syntax highlighting definitions now independant of editing mode.
+ Event mappings now independant of editing mode.
+ Abbreviatons. Can expand the text or run a macro.
+ Searching can now check for words without using regular expressions.
+ Loading files is now almost twice as fast.
+ File positions and prompt history is now saved in file FTE.HIS.

+ The list of loaded files is saved on exit to FTE.DSK. Files are automatically loaded on startup. Several settings and command line options are available to configure this.
+ On startup, only the first file is actually loaded. Other files are loaded only as they are needed.
+ BlockTrans, CharTrans and LineTrans commands. Can translate characters according to arguments (BlockTrans 'a-z' 'A-Z', etc).
+ When cursor is over the bracket, the matching bracket can be higlighted automatically of visible on screen.
! InsPrevLineChar, InsPrevLineToEol failed when tabs were on previous line.
+ Nonpersistent blocks (with various options and commands).
+ …

0.37—Dec 95

- Status line can now be hidden. Also changed it's look.
- Changed the syntax of keyboard bindings. Now it is possible to better emulate the wordstar two-key behavior. See documentation for details. It is also possible to define different commands for gray/white keys.
! MENU shortcuts now work.
+ New command: FileWriteTo
! Fixed crash when trying to center nonexistant line in file.
- Changing folds now modifies the file.
- When CursorTroughTabs was set to 0, certain movement commands would behave incorrectly.
+ S-Ins will perform Paste operation in prompts.
+ New Command: WinResize <delta> and WinClose. Windows can be also resized by a mouse.

0.36—Oct 95

! Fixed when editor would crash when saving a folded file, but no folds are configured for active mode.
! Fixed minor bug in C mode indentation.
+ New command: InsertSpacesToTab (takes optional tabsize argument).

0.35—Oct 95

KillWordPrev now works correctly.
FindReplace command works correctly if WSStyleSearch == 1.

0.34—1995/10/15

Minor fixes & docs updates…

0.33—1995/10/01

! SIGBREAK handler now works again.
! Fixes in C/C++ smart indentation (if in switch, …)
+ Pascal highlighting mode.

+ Printing.
+ Rewritten folding. Now supports nested folds, opening, closing folds, and persistent folds.
+ New folding commands: FoldCreate, FoldDestroy, FoldOpen, FoldClose, FoldPromote, FoldDemote, FoldCreateByRegexp, FoldOpenAll, FoldOpenNested, FoldCloseAll, FoldDestroyAll, FoldToggleOpenClose, MoveFoldPrev, MoveFoldNext.
+ New settings: SaveFolds, CommentStart, CommentEnd.
+ Word characters can be configured using WordChars setting.

0.32—1995/08/15

+ New search routines. (Find, FindReplace, FindRepeat, FindRepeatReverse, FindRepeatOnce, …)
+ Block-local searches.
+ In buffer-list, most recently used files will now be listed first.
+ Main menu bar can now be hidden.
+ Performance improvements.
+ Bookmarks! New commands: PlaceBookmark, GotoBookmark, RemoveBookmark
+ Files can now be saved and closed from window list.
+ Optimized CMode indentation. Also more configurable.

0.31—1995/07/31

! Bug fixed in undo/redo when UndoLimit reached.

0.30—1995/07/30

+ Folding support.
+ New commands: FoldLine, UnfoldLine, UnfoldNextLine, UnfoldAll, ClearFolds, FoldIndent, FoldRegexp, FoldBlock, UnfoldBlock, FoldBlockRegexp.
! BackSpace at eof when TrimLine is enabled will not abort.
+ Incremental search (IncrementalSearch).
+ PgUp/PgDn on a file prompt will show a list of files.
+ New command: CompleteWord

0.29—1995/07/20

Regexps can now be case insensitive (\C,\c).
BlockRead/BlockReadColumn/BlockWrite commands.
Block marking can now be undone.
Commands that prompt for string/int values can now take string/int arguments.
Multiple compile-regex statements can be specified simultaneously
Minor bug-fixes in regexps ([\x00-\xFF] now works).
Filter for filename completion.
New commands: SwitchTo ChangeKeys ChangeFlags ShowMenu
New options: CompletionFilter DefaultModeName

0.28—1995/07/08

- Needs to have documentation updated.
Mostly rewritten PERL highlighting. Works much better now.
Completely new config file syntax.
New commands: ASCIITable, LoadFileInMode
CMode indentation should now work for Perl (close enough).
Highlighting for ADA and Email messages.

0.27—1995/06/19

Minor bug fixes.

0.26—1995/06/18

New commands: {Char,Line,Block}Case{Up,Down,Toggle}
New setting: LoadAfterQuit—if set to 1, editor will prompt to load another file before exiting.
New setting: ShowScrollBar {0,1}.

0.25—1995/06/12

Minor bug fix in REXX highlighting ("\"", …)
Bug fixes in word wrap.
BlockCut now doesn't move the cursor to the block beginning.
New commands: MoveLineTop, MoveLineCenter, MoveLineBottom.
Editor will now check if the file has changed before the first modification.
Found text is now highlighted.
New CMode setting: C.BraceOfs and command: ChangeCBraceOfs
New commands: MovePrevPos, SavePos, MoveSavedPos, MoveSavedPosCol, MoveSavedPos-Row

0.24—1995/06/06

When checking for file modification time of last change is now used instead of the time of last access.
Wildcard support for file loading.

0.23—1995/06/04

Ctrl+C and Ctrl+Break are now disabled.
Ctrl+S and Ctrl+C keys are now again recognised in Windowed mode.
Fixed problem when spawning a subprocess in Windowed mode.
New command: ShowEntryScreen
ListRoutines in CMode only shows functions not their prototypes.
New setting: SysClipboard - if set to 1, editor will automatically use system clipboard.
New command: ToggleSysClipboard.

Minor bug fix in PM clipboard support.
New commands: BlockPasteStream, BlockPasteColumn and BlockPasteLine. BlockPaste command will now always paste in current block mode, not in the last Copy/Cut mode.
More than 4 commands can be bound to a key (actually this worked since 0.18, but was not documented).
New command: FileReload.
Editor will now check if file has changed on disk before saving it.

0.22—1995/05/28

But fix in regular expressions (nested +#*@).
Changes in regular expression syntax.
New function: ListRoutines. Shows functions in current buffer.
New setting: RoutineRx

0.21—1995/05/21

REXX mode smart indentation.
KillWord & KillWordPrev commands now actually work.
Pressing Ctrl+Enter to begin Search will toggle case sensitivity of search.
New option 'Trim' and commands 'ToggleTrim', 'LineTrim'. Removes whitespace from end of lines.
New option 'ShowMarkers' and command 'ToggleShowMarkers'. Shows end of line and end of file markers.
Bug fix in PERL highlighting (caused lockups)
Bug fixes and improvements in regular expressions.

0.20—1995/05/18

Major bug fixes in word wrap.
New commands: MovePrevTab, MoveNextTab.
Bug fixes in BlockIndent and BlockUnindent (stream/line mode)

0.19—1995/05/16

Function names in REXX are now highlighted.
WordWrap can be set to 0 - disabled, 1 - wrap line at right margin and 2 - wrap paragraph continously. Function ToggleAutoWrap renamed to ToggleWordWrap.
New way to set left/right margin (SetLeftMargin, SetLeftMargin)
Minor fix in PERL highlighting.

0.18—1995/05/13

PERL Syntax Higlighting.
Memory allocation problem in tab expansion.
Wordwrap now strips all spaces on beginning of line (except on the first line of the paragraph).

Tabs can be set to any number between 1 and 32.
Changed names of buffer flags (WrapOn -> AutoWrap, UndoRedo -> Undo, ShowTab -> ShowTabs)
New commands: ToggleAutoIndent, ToggleExpandTabs, ToggleShowTabs, ToggleUndo, ToggleReadOnly, ToggleKeepBackups, ToggleMatchCase, ToggleBackSpKillTab, ToggleDeleteKillTab, ToggleSpaceTabs, ToggleIndentWithTabs, ToggleBackSpUnindents, ToggleAutoWrap.
New commands: WinRefresh, ChangeTabSize, ChangeCIndent, ChangeLeftMargin, ChangeRightMargin.

0.17—1995/05/10

Minor fix in word-wrap.
Screen repaint problems when shelling out fixed.
Blinking disabled for full-screen, high-intensity background colors can now be used.

0.16—1995/05/06

Minor speedups in screen handling and highlighting.
Regular expressions can now match start and end of words using .
Regexp replace can paste entire matched string using &.
Regexp search/replace could match part of just replaced string.
New commands: MoveLastNonWhite, MovePrevEqualIndent and MoveNextEqualIndent, LineDuplicate, InsPrevLineChar, InsPrevLineToEol.
New color config. 'C.Function' for functions in C highlighting mode.
Improved CMode hilit, preprocessor hiliting improved (strings, comments, numbers)

0.15

1995/04/29

Speed improvement in CMode auto indent.
Delete command can now delete full tabs instead of converting them to spaces.
When closing a modified file, editor prompts you to save it.
Automatic indentation can now use tabs.
Manual and automatic wordwrap.

1995/04/24

Backspace can now delete full tabs instead of converting them to spaces
(See BackSpace and KillBackTab).
Backspace can unindent to previous indentation level.

1995/04/20

Basic mouse support.

1995/04/12

Configurable colors/keywords in C/REXX mode.

0.14—1995/04/07

Characters could not be entered using AltGr on international keyboards.
Immediatelly doing an undo on a newly loaded file deleted the first line.
Ascii characters >= 128 can be now entered without quote command (C-Q).
IPF Syntax highlighting.

0.13—1995/04/03

Bug fix in regular expressions.
Bug fixes in compiler support
Editor clipboard can now be copied to/from PM clipboard.

0.12

1995/03/30

Compiler support + error message parsing

1995/03/25

Paren matching (Command: MatchBracket).
Bug fixes in CMode smart indentation.

0.11

1995/03/11

Unlimited undo now works again.

1995/03/18

Entire blocks of C code can now be reindented (BlockReIndent)
Search can now be case insensitive (SearchMatchCase - toggle). Option: MatchCase, Command: SearchMatchCase
Regular expression find/replace works (case sensitive only)

1995/03/19

Fixed a bug in redo (last command could not be undone)
New option: KeepBackups—if set to 0, backup files will be deleted after a succesful save.

0.10

1995/03/06

Fixed CMode indent when tabs are present in the file.
Prompts now retain previous text only if you try to edit it.

1995/03/04

4DOS/4OS2 style filename completion (FileOpen, …).

1995/02/25

New load routine, much faster in some cases.
Undo/Redo can now be limited (if you hate to waste memory).

1995/02/19

C Mode indentation level can now be specified (C.Indent)
Bug fixes in screen redraw.
Editor will now scroll text instead of always redisplaying the screen.
Regular expressions (Search only).

0.09—1995/02/08

First public release (Version 0.09b)