jade

COLLABORATORS

	TITLE : jade		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		September 19, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

jade

1.1 jade.guide

```
                Jade
****

   Jade is a highly flexible text editor for the Unix (with X11) and
Amiga operating systems.

   This is Edition 1 of its documentation, last updated 19 April 1994
for Jade version 3.0.
```

1.2 jade.guide/Copying

Copying
****

Jade is distributed under the terms of the GNU General Public License, this basically means that you can give it to anyone for any price as long as full source code is included. For the actual legalese see the file ‘COPYING’ in the distribution. I reserve the right to use a different license in future releases.

The only parts of Jade which are not my own work are the regexp code, this is by Henry Spencer (though I have made some small modifications) and is distributed under his conditions, and the ARexx interface in the Amiga version which is based on ‘MinRexx’ by Radical Eye Software.

Be aware that there is absolutely NO WARRANTY for this program, you use it at your own risk. Obviously I hope there are no bugs, but I make no promises reguarding the reliability of this software.

1.3 jade.guide/Introduction

Introduction
****

Jade is a text editor primarily designed for programmers. It is easily customised through a Lisp-style extension language and can be tailored to the user’s own requirements.

Jade is designed to run under a graphical windowing system, systems currently supported are the Commodore Amiga and the X Window System version 11 (but only under Unix).

It is the successor to the editor ‘Jed 2.10’ which I released for the Amiga in early 1993. I have decided to rename it now that I have made a

Unix version since there is already an editor called 'Jed' available (there is no connection between the two, I haven't even looked at the other one). "Jade" is an anagram of "A Jed", if you want an acronym you could use "Just Another Damn Editor", if you can think of anything better please tell me.

Jade is compatible with GNU Emacs in terms of keystrokes and command names to a certain extent but it is not intended as a simple copy of Emacs (indeed, when I started this I had never actually used Emacs!). I have tried to take my favourite aspects of all the editors I have used as well as adding features that I have not found elsewhere. Consequently, it is very much the editor that *I* want -- you may not find it so appealing.

The feature that the most people will dislike is that it doesn't support "proper" tabs. By this I mean that it expands all tabs to a sequence of spaces when loading a file, they are not converted back to tabs until the file is saved back to disk (though this is optional).

1.4 jade.guide/Systems Supported

Requirements
****

Jade will only run on certain operating systems, this chapter details just what it needs as well as some notes relevant to each system.

1.5 jade.guide/Amiga Jade

Amiga Jade
==========

The only real requirement for Jade running on an Amiga is that it must run an operating system revision of at least V37 (thats V2.04) and have about 300K free memory available.

It also needs more stack than the average Amiga application. For normal use 20K should be okay. If you want to use the Lisp compiler 50K would be a better bet.

It assumes that its directory is pointed to by the 'JADE:' assignment. This means that the main Lisp files are stored in 'JADE:lisp/' and the file of doc-strings is 'JADE:DOC-strings'.

1.6 jade.guide/Unix and X11 Jade

Unix and X11 Jade
=================

Jade will only run on version 11 of X, it has absolutely no support for character terminals or different windowing systems. As long as it compiles it should work on your system.

One problem you might find is that the BackSpace and Delete keys don't work properly. As far as I have been able to find out, most X terminals map both the BackSpace (normally at the top-right of the alpha-keyboard) and the Delete (normally somewhere above the cursor keys) keys to the 'Delete' keysym. Obviously, since I want these keys to have different effects (1) this is no good. What I decided to do about this was two things,

1. Use 'xmodmap' to map the Delete key to the 'BackSpace' keysym. This may sound backwards but most programs seem to use the 'Delete' keysym as what I call 'BackSpace' so mapping as I described doesn't break this.

   To do this, I have the following in my '.Xmodmap' file

   ```
   keycode 107 = BackSpace
   ```

   Note that the '107' is the Delete key's keycode on *my* keyboard, your keyboard may, and probably will, be different.

2. In the function which binds descriptions of keystrokes to Lisp forms, swap the meanings of the 'BackSpace' and 'Delete' keysyms.

This means that everything works okay! You can bind to Delete key and it will work properly.

---------- Footnotes ----------

(1) BackSpace should rub out the key before the cursor and Delete should delete the character under the cursor

1.7 jade.guide/Editor Concepts

Editor Concepts
****

Before I describe the editor in detail there are several concepts which you should be familiar with. Some will be explained in more detail later.

"buffer"
Buffers are used by the editor to store the text that you are editing. Broadly speaking, each buffer holds the contents of one text-file loaded into the editor (it is not necessary for each buffer to be associated with a file, some buffers exist for other purposes for example the '*jade*' buffer is used to interact with the Lisp system.

"current buffer"
The buffer being edited in the current window (see below), most editor commands work on this buffer unless told otherwise.

"window"
Corresponds to a window in the window-system. Each window can display one buffer at a single time (although a buffer may be displayed in more than one window at once).

"current window"
Jade always keeps track of which one of its windows is active. It is called the current window. Whenever you type a key or press a mouse button in one of Jade's windows, that window automatically becomes the current window. Amongst other things, all messages from the editor are displayed in the status line of the current window.

"cursor"
The cursor marks your current position in the current buffer (see above), when you type something it is inserted into the buffer between the cursor and the character preceding it (unless you type a command).

"status line"
One line in a window is devoted to displaying messages from the editor,
Using Windows
.

"Lisp"
The programming language which Jade uses, although the internals of the editor are written in C, all commands are written in a dialect of Lisp (even if the command only calls a C function). Jade contains an interpreter, compiler and debugger for this language. See
Programming Jade
.

"variable"
Variables are used to store Lisp values, each variable has a unique name. Note that unlike many programming languages variables in Lisp are *not* typed, the data values themselves have a type associated with them.

"form"
A form is a single Lisp expression. For example, all of these are forms:

```
foo
```

```
          42
          "hello"
          (setq foo 200)
```

"command"
     A command is a sequence of Lisp forms which may be called interactively (ie, from the keyboard). It may be a key sequence (such as ‘Ctrl-x Ctrl-f’) or a Lisp function to evaluate (such as ‘ESC x find-file’).

"regular expression"
     A regular expression is a string which is used to match against other strings. It has a special syntax which allows you to form a kind of template against which the other strings can be matched. They are used extensively by the editor, but you -- the user -- will mainly encounter them when searching and replacing strings in buffers.

1.8 jade.guide/Key Definitions

Key Definitions
****

In this manual I have adopted a consistent notation for all keypresses, since most editor commands are invoked via a typed key-sequence it is very important that you can decipher this notation.

Every keypress has a set of "modifiers"; these are the keys such as "Shift" or "Control" which don’t actually produce a character when typed, they only effect the rest of the keyboard. Each key, then, can have one or more modifiers.

The actual key definition consists of zero or more hyphen-separated modifiers, followed by a hyphen and the name of the actual key (or event).

Some commands are triggered by a sequence of one or more of these key definitions, press each key definition in turn to invoke the command.

Note that the case of modifiers is not important, however some of the keys *are*, so you should always specify them in their correct case.

1.9 jade.guide/Modifiers

```
Modifiers
=========

"Shift"
"SFT"
     The shift key.

"Ctrl"
"CTL"
     The control key, or its equivalent.

"Meta"
     This depends on the window-system, on X11 it is the "Mod1"
     modifier, on the Amiga the "Alt" key.

"LMB"
     The left mouse button.

"MMB"
     The middle mouse button.

"RMB"
     The right mouse button.

   As well as these, there are also some others, "Mod1" to "Mod2"
represent the X11 modifiers of the same name. "Button1" to "Button5"
also correspond to their X11 conterparts (Button1 to Button3 are LMB to
RMB). For Amiga users, "Amiga" corresponds to the Amiga key (this is
the same as Mod2).
```

1.10 jade.guide/Keys

```
Keys
====

   As far as possible each single character key-definition corresponds
to where that character is on the keyboard (a is ‘a’, etc...).

   When using an Amiga this should be true for *all* keys since the
Amiga’s "keymap.library" makes it easy to look up what key a character
belongs to.  However, this is not so easy on X11. All of the standard
ASCII character set should be okay, but the more esoteric characters
may have to be specified by the names of their X11 keysym (without the
‘XK_’ prefix). Look in the <X11/keysymdef.h> include file for all
keysyms, for example ‘XK_question’ would have to be used for ‘?’ if the
editor didn’t treat it, and many others, specially.
```

Some keys which don’t follow this pattern are

"Space"
"SPC"
"SpaceBar"
: The space bar.

"TAB"
: The tab key.

"RET"
"Return"
: The return key.

"ESC"
"Escape"
: The escape key.

"BS"
"BackSpace"
: The backspace key.

"DEL"
"Delete"
: The delete key.

"HELP"
: The help key, not all keyboards have this.

"UP"
: The cursor up key.

"DOWN"
: The cursor down key

"LEFT"
: The cursor left key.

"RIGHT"
: The cursor right key.

"KP_Enter"
"KP_Multiply"
"KP_Divide"
"KP_Minus"
"KP_Add"
"KP_Decimal"
"KP_N"
: Keys on the numeric keypad. For KP_N, N is a digit.

"Click1"
: Single clicking a mouse button.

"Click2"
: Double clicking a mouse button.

"Off"

```
     Releasing a mouse button.

"Move"
     Moving the mouse. This doesn’t work on X11 yet.
```

1.11 jade.guide/Example Keys

```
Example Keys
============

   Some examples of proper key definitions are,

‘Ctrl-x’
     Hold down Control, type x.

‘Meta-Shift-RET’
     Hold down Meta and Shift, then type the Return key.

‘LMB-Click1’
     Click the left mouse button once.

‘Meta-RMB-Click1’
     Hold down Meta then click the right mouse button once.
```

1.12 jade.guide/Starting Jade

```
                Starting Jade
*************

   This chapter describes Jade’s initialisation process. This includes
how to start it, what options it will accept and what it actually does
after being started.
```

1.13 jade.guide/Invocation

```
                Invocation
==========

   Since Jade supports two vastly different operating systems they both
need to be covered separately.

   * Amiga

     The normal way to start Jade on the Amiga is to type its name at
     the Shell (or CLI) together with any options (see
                Startup Options
                )
     you want. Note that these options are in the traditional Unix
     style, a dash followed by the option name and any arguments, not
     the standard AmigaDOS method.

     It is also possible to invoke the editor from the Workbench,
     simply double clicking on its icon will cause Jade to open its
     initial window. Unfortunately there is no support for passing
     arguments via Tool Types, nor is there any way to create icons
     with saved files. This is largely due to the fact that I rarely
     use the Workbench -- if enough people complain about this I will
     probably fix it. Jade doesn't have an icon yet, you'll have to
     make one yourself.

   * Unix with X11

     Jade should be started like most other Unix programs, type its
     name and any arguments to a shell. It must be able to connect to
     an X server (preferably the one controlling your terminal), the
     `-display' option can be used if needed.
```

1.14 jade.guide/Startup Options

```
Startup Options
===============

   The acceptable options can be split into three classes. Note that
they must be specified on the command line in order of their class.
This means that, for example, the `-rc' option must be after the `-font'
option.

   So, the general usage pattern is

     jade [SYSTEM-DEPENDANT-OPTIONS] [STANDARD-OPTIONS] [LISP-OPTIONS]

   Note that the LISP-OPTIONS may include files to be loaded.

  1. System dependant options.

          * Options for the Amiga system.
```

```
      `-pubscreen SCREEN-NAME'
           Defines the name of the public screen on which the first
           window is opened. By default (or if SCREEN-NAME doesn't
           exits) the `Workbench' screen is used.

      `-font FONT-STRING'
           Defines the font used in the first window. FONT-STRING
           is the font to use, it is the name of the font (for
           example, `topaz.font'), followed by a hyphen and the
           point size to use. For example, a FONT-STRING of
           `topaz.font-8' gives 8-point topaz. This is the default.

    * Options for X11.

      There are two types of options to the X11 version of the
      editor, those specified on the command line and those defined
      in the resource database (ie, in your `.Xdefaults' file).
      Resources are looked for under two names, firstly the name
      under which the editor was invoked (normally `jade'), if this
      fails it tries again with the name `Jade'. Naturally, options
      specified on the command line overide those in the resource
      database.

      `-display DISPLAY-NAME'
           Defines the name of X display to open, by default the
           contents of the environment variable `DISPLAY'. It is a
           string of the form `hostname:number.screen_number'.

      `-name NAME'
           The name to use when looking up resource values, this
           replaces the base name of the executable (normally
           `jade').

      `-geometry GEOM-SPEC'
           Specifies where to place the first window on the screen.
           This is a standard X style geometry specification.

      `-fg FOREGROUND-COLOUR'
      `fg: FOREGROUND-COLOUR' *RESOURCE*
           The colour of the window's foreground (ie, the text).

      `-bg BACKGROUND-COLOUR'
      `bg: BACKGROUND-COLOUR' *RESOURCE*
           The background colour of the window.

      `-font FONT-NAME'
      `font: FONT-NAME' *RESOURCE*
           The name of the font used for all text in the initial
           window.

  2. Standard options.

    `-rc LISP-FILE'
         Load the Lisp script LISP-FILE instead of the normal
         initialisation script (`init'). Warning: the editor depends
         heavily on the normal file, if you change this without due
         care the editor could be unusable -- no keys will be bound
```

```
          and many standard functions won't exist.

    `-v'
          Print the version and revision numbers of this copy of the
          editor then quit.

    `-log-msgs'
          This option makes all messages which are displayed in the
          status line also be written to the standard error stream.
          This is sometimes useful for debugging purposes.

  3. All other options are passed to the Lisp initialisation process in
     the variable `command-line-args', these are available to any Lisp
     packages loaded in the initialisation script. Any left after that
     are scanned for the following options,

    `-f FUNCTION'
          Call the Lisp function FUNCTION.

    `-l FILE'
          Load the Lisp file FILE.

    `-q'
          Quit cleanly.

    `FILE'
          Load the file of text FILE into a new buffer.

   An example command line for starting Jade from a Unix shell could be

     $ jade -fg white -bg black -log-msgs foo.c bar.jl

   This means white text, black background, save messages and load the
files `foo.c' and `bar.jl'.
```

1.15 jade.guide/Startup Procedure

```
Startup Procedure
=================

   This is a description of what happens when the editor initialises
itself.

  1. Firstly lots of internal data structures are created, memory
     pools, symbols and their symbol-table (including all the primitive
     Lisp functions).

  2. The window-system is initialised (parse the system-dependant
     options, and the xrdb resources if in X).

  3. Parse the standard options.

  4. Create the initial window and the first buffer to display in it
     (this is the buffer called `*jade*').
```

5. Load the initialisation script, this is either the Lisp file called 'init' or whatever was given to the '-rc' command line option.

   Some selected highlights of what the standard file does are,

   * Load lots of Lisp files, some notable ones are

     'loadsyms'
          Initialise the autoload stubs.

     'loadkeys'
          Creates the standard keymaps and keybindings.

   * Try to find the user's personal startup file, this is normally the file '.jaderc' in their home directory (1).

   * Load any files which were specified on the command line.

6. Start the top-level recursive edit, this doesn't exit until the editor does.

---------- Footnotes ----------

(1) The Amiga has no notion of a user's home directory, Jade uses the contents of the environment variable 'HOME', or if this doesn't exist the 'SYS:' assignment.

1.16 jade.guide/Using Jade

Using Jade
****

This chapter of the manual is meant to teach you to *use* the editor, because of this I have attempted to reduce references to the Lisp extension language to an absolute minimum. The only things that you need to know about at this time are how to set or reference Lisp variables and how to invoke Lisp commands.

Luckily, it is very easy to do this by typing one of the following sequences to the editor,

'ESC x FUN'
     Calls the command called FUN and displays its result in the status line.

'ESC x set-variable RET FOO RET BAR'
     This sets the Lisp variable FOO to the value BAR.

'ESC x show-variable RET FOO'
     This displays the value of the variable FOO in the status line.

Note that throughout this manual it is assumed that you press the RET

```
(return) key after each 'ESC x' command. For example, to invoke the
command 'ESC x find-file' you would actually type the following (but
not the spaces -- they are for readability).

     'ESC x f i n d - f i l e RET'
```

1.17 jade.guide/The Help System

```
                The Help System
===============

   To invoke the help system type the key sequence `Ctrl-h' or if your
keyboard has it the `HELP' key.

   A prompt will be displayed in the status line showing you which keys
you can press next to enter one of the main options of the help system
explained below. Alternatively, you can type either `Ctrl-h' or `HELP'
again to display some text telling you more about the help system and
how to use it.

   The help system is exited after successfully invoking one of the
commands described below or typing anything which is not a recognised
command to the help system.

`a'
     To list all function names matching REGEXP, type `a REGEXP RET'
     when in the help system.

`e'
     Similarly, to list all variable names matching REGEXP, type `e
     REGEXP RET' when in the help system.

`f'
     Displays the online documentation for a function. After invoking
     this option type the name of the function.

`h'
     Shows some helpful text describing how to use the help system.

`i'
     Enters the Info viewer. This allows you to browse through files
     written in the Info hypertext format. For more information see

                Info-mode
                , for more information on Info files in general see Info.

`m'
     Display the current editing modes documentation.

`v'
     Displays the online documentation and current value of a variable.
     Type the name of the variable after invoking this option.
```

1.18 jade.guide/Loading and Saving Files

```
                Loading and Saving Files
========================

   Since `jade' is a text editor its main function is to edit files of
text.  This means that you must be able to read the text contained in a
file into one of the editor's buffers, then save it back to disk when
```

you have finished editing it. That is what this section deals with.

1.19 jade.guide/Commands To Load Files

Commands To Load Files

There are several commands used to load files into buffers, these are,

'Ctrl-x Ctrl-f'
Prompts for the name of a file (using file-completion) and display the buffer containing that file. If the file has not already been loaded it will be read into a new buffer.

'Ctrl-x Ctrl-v'
Prompts for the name of a file, the current buffer is killed and the buffer in which the prompted-for file is being edited is displayed. As in 'find-file' it will be read into a new buffer if it is not already in memory.

'Ctrl-x Ctrl-r'
Similar to 'find-file' except that the buffer is marked as being read-only. This means that no modifications can be made to the buffer.

'Ctrl-x i'
Prompts for a file, then inserts it into the current buffer at the cursor position.

You can use the prompt's completion feature to expand abbreviated filenames typed to the prompt, for more information see

The Buffer Prompt
.

1.20 jade.guide/Commands To Save Files

```
Commands To Save Files
----------------------
```

These are the commands used to save buffers and the keystrokes associated with them,

'Ctrl-x Ctrl-s'
: Saves the current buffer to the file that it is associated with (this is either the file that it was loaded from or something else set by the function 'set-file-name'). If no modifications have been made to the file since it was loaded it won't be saved (a message will be displayed warning you of this).

'Ctrl-x Ctrl-w'
: Prompts for a name to save the file as. The file associated with this buffer is renamed and the file is saved as its new name.

'Ctrl-x s'
: For each buffer which has been modified since it was loaded, ask the user if it should be saved or not. If so, the command 'save-file' is used to save the file

1.21 jade.guide/Backup Files

```
Backup Files
------------
```

The editor can optionally preserve the previous contents of a file when they are about to be overwritten by saving a buffer. It does this by renaming the old file, 'foo' as 'foo~' (the original name plus a tilde appended to it).

- Variable: make-backup-files
  This variable controls whether or not backups are made of files about to overwritten by the function 'write-buffer' (ie, the commands 'save-file' and 'save-file-as'). When non-nil the old instance of the file is renamed so that it has a tilde appended to its old name.

- Variable: backup-by-copying
  When non-nil all backups are made by copying the original file

instead of renaming it as the backup file. This is slower but less destructive.

If ‘backup-by-copying’ is nil and renaming the original file as its backup would be damaging (ie, changing the ownership of the file or breaking a link) no backup will be made.

1.22 jade.guide/Auto-Saving Files

Auto-Saving Files

Jade is able to save snapshots of a buffer’s contents at set time intervals. When this time interval expires and the buffer has been modified since it was last (auto-) saved to disk (and the editor is idle) the buffer is saved to a special file (usually the base component of the file’s name surrounded by ‘#’ characters in the file’s directory).

- Variable: auto-save-p

  When non-nil this makes the function ‘open-file’ (and therefore the commands ‘find-file’, etc) flag that the file it just read should be auto saved regularly.

- Variable: default-auto-save-interval

  This is the default number of seconds between each auto save. This variable is only referenced when each file is opened.

  Its standard value is 120 seconds.

- Variable: auto-save-interval

  This buffer-local variable controls the number of seconds between each auto-save of the buffer it belongs to. A value of zero means never auto-save.

When the buffer is saved properly (ie, with ‘save-file’ and friends) its auto-save file is deleted. Note that this doesn’t happen when you kill a buffer and an auto-save file exists (in case you didn’t mean to kill the buffer).

If you want to change the format of the name of auto-saved files look at the function ‘make-auto-save-name’ and its documentation.

1.23 jade.guide/Loading and Saving Tabs

Tab Expansion

The editor does not leave tab characters (ASCII 9) as they are. They are expanded into one or more spaces when the file is read into its

```
buffer. The size of the expansion depends upon the column number at
which the tab has occured and the value of the variable `disk-tab'.

 - Variable: disk-tab
     This buffer-local variable determines the size of tab stops used
     when a file is read from disk (by the `read-buffer' function) into
     a buffer.

   It is desirable that the files which the editor produces have tabs
in them though, so something has to be done.

   The variable `save-tabs' controls exactly how files are saved (in
respect to saving tab characters).

 - Variable: save-tabs
     The value of this buffer-local variable is used to decide exactly
     which sequences of spaces are changed to tab characters when a
     buffer is saved to disk (with the `write-buffer' and
     `write-buffer-area' functions).  There are three possible options
     (all of which are Lisp symbols):

    `nil'
          *No* tabs are saved at all. All whitespace is left untouched,
          this may be necessary with some types of file whose format is
          strongly defined.

    `leading'
          Any whitespace at the start of each line is translated into
          tabs and spaces such that the first non-whitespace character
          in the line is at the same logical position as it was in the
          buffer.

    `all'
          Any sequence of two or more space characters is translated
          into tab characters when the logical structure of the line
          would be unaltered by doing so. *No* translations take place
          after the first quote character (ie, `'', `'' or `"') in the
          line (this is to try and prevent errors).
```

1.24 jade.guide/Automatic Mode Selection

```
Automatic Mode Selection
------------------------

   As described elsewhere in this manual, each buffer can have an
editing mode associated with it (ie, `c-mode' for editing buffers of C
source code).

   Since it would be extremely tedious to have to invoke the mode's
initialisation function manually whenever a new file is loaded the
editor can initialise the mode automatically. It does this by scanning
an association list called `mode-alist' for a regular expression
matching the name of the file (or the string in the buffer-local
variable `mode-name' if it is non-nil). If a match is found the function
```

associated with the matching regular expression is called, thereby initialising the mode.

If you don't understand this, don't worry -- it works.

For example, the mode-alist contains this fragment as one of its elements:

```
     ("\.[ch]$" . c-mode)
```

which means call the function `c-mode' for any file ending in `.c' or `.h'.

```
 - Variable: mode-alist
     A list of elements of `(MATCH-REGEXP . MODE-FUN)'. When a file is
     loaded each MATCH-REGEXP is compared with the name of the file in
     question (or it's `mode-name' value). When a match is found the
     corresponding MODE-FUN function is called.
```

1.25 jade.guide/Embedding Lisp In Files

Embedding Lisp In Files

It is possible to include Lisp commands in the text of a file so that they will be read and evaluated when that file is loaded into a buffer.

This is normally used to set buffer-local options which are specific to one particular file, ie, to set the name of the editing mode, or the size of tab characters in this file.

The way to do this is to include a section of text of the form in the file:

```
     ...
     XXX ::jade-code::
     XXX Lisp Line1
     XXX Lisp Line2
     XXX ...
     XXX Lisp LineN
     XXX ::end::
     ...
```

The `XXX' just means that any text to the left of the column in which the `jade-code' begins is ignored (This is mainly to allow for any comments needed to make sure that the Lisp text is not used by whatever uses the file).

Only one block such as this is allowed per file, it is not evaluated until the whole of the file has been read.

Some examples uses of this could be,

```
   In a lisp file:
     ;;; ::jade-code::
     ;;; (setq lisp-mode-tab 4)
     ;;; (setq mode-name "lisp-mode")
     ;;; ::end::

   Or in a C source file:
     /* ::jade-code::
      * (setq c-mode-tab 4)
      * (setq mode-name "c-mode")
      */::end::
```

It is also possible to prohibit the evaluation of these special sections.

 - Variable: no-file-code-p
     When non-nil the section of a file marked for auto-evaluation (with the `::jade-code::' marker) is *not* evaluated.

1.26 jade.guide/Editing Buffers

Editing Buffers
===============

The majority of keys when typed will simply insert themselves into the buffer (this is not always true but it's a good assumption) since they have not been bound. Typically this includes all normal characters (ie, alphanumeric, puntuation, etc) as well as any of the more obtuse key-sequences which have not been bound to a function (`Ctrl-l' is one of the more useful of these).

The behaviour of the TAB key is different to many other editors -- it doesn't insert anything (unless a specific editing mode has bound it to something else, like `c-mode' for example), generally it just moves the cursor to the next tab stop. This is partly because Jade doesn't use "proper" tabs and partly because it makes it easier to move around a line (because the keystroke `Shift-TAB' moves to the previous tab stop).

Some miscellaneous editing commands are,

`RET'
     This generally splits the line into two at the position of the cursor, some editing modes may provide an option which automatically indents the line after it's split.

`BS'
     Deletes the character before the cursor.

`DEL'
     Deletes the character under the cursor.

`Shift-BS'
     Kills the characters between the start of the line and the cursor.

'Shift-DEL'
'Ctrl-d'
     Kills the characters from the cursor to the end of the line.

'Ctrl-DEL'
     Kills the whole line.

'Ctrl-o'
     Splits the line in two at the cursor, but leaves the cursor in its original position.

'ESC d'
'ESC DEL'
     Kills from the cursor to the end of the current word.

'ESC i'
     Inserts spaces to fill from the cursor to the next tab stop.

'ESC l'
     Makes the characters from the cursor to the end of the word lower case.

'ESC u'
     Upper cases the characters from the cursor to the end of the word.

'ESC BS'
     Kills from the cursor to the beginning of the word.

1.27 jade.guide/Moving Around Buffers

```
Moving Around Buffers
=====================
```

These are the commands which are used to move the cursor around the current buffer,

'UP'
'Ctrl-p'
     Move one line up.

'DOWN'
'Ctrl-n'
     Move one line down.

'LEFT'
     Move one column to the left, stopping at the first column.

'Ctrl-b'
     Move to the previous character, at the beginning of the line moves to the end of the previous line.

'RIGHT'
     Move one column to the right. This keeps moving past the end of the line.

`Ctrl-f'
: Move to the next character, at the end of a line moves to the start of the next line.

`Shift-UP'
: Move to the first line in the buffer.

`Shift-DOWN'
: Move to the last line in the buffer.

`ESC <'
: Move to the first character in the buffer.

`ESC >'
: Move to the last character in the buffer.

`Shift-LEFT'
`Ctrl-a'
: Move to the beginning of the current line.

`Shift-RIGHT'
`Ctrl-e'
: Move to the last character in the current line.

`Ctrl-UP'
`ESC v'
: Move to the previous screenful of text.

`Ctrl-DOWN'
`Ctrl-v'
: Move to the next screenful of text.

`Meta-LEFT'
`ESC b'
: Move to the previous word.

`Meta-RIGHT'
`ESC f'
: Move to the next word.

`Meta-UP'
`ESC ['
: Move to the start of the previous paragraph.

`Meta-DOWN'
`ESC]'
: Move to the start of the next paragraph.

`TAB'
`ESC TAB'
: Move to the next tab position. Note that some editing modes redefine TAB to make it indent the current line.

`Shift-TAB'
: Move to the position of the previous tab.

```
`Ctrl-TAB'
`ESC i'
     Insert a tab (ie, enough spaces to move to the next tab position).

`Ctrl-j'
`ESC x goto-line'
     Prompt for a line number and go to it.

`ESC m'
     Move to the first non-space character in the current line.

   There are several variables which affect the commands described
above, these are,

 - Variable: screen-tab
     This buffer-local variable controls the size of tab characters in
     a buffer.  Its standard value is 8. This variable does not affect
     the size of tabs in files read into the buffer, that is controlled
     by `disk-tab'.

 - Variable: y-scroll-step-ratio
     A window-local variable which controls what happens when you move
     the cursor off the top or bottom of the window. A value of zero
     means move as much as needed to get the cursor back into view, for
     example, if you move down one line, it will scroll the window one
     line only. If the value is not zero the screen is moved by the
     number of rows in the window divided by the value. For example, a
     value of 2 means scroll the window in chunks half the size of the
     window -- this is useful for when you are working with a slow
     updating display.

 - Variable: x-scroll-step-ratio
     Similar to `y-scroll-step-ratio' except for horizontal movement.
```

1.28 jade.guide/Cutting And Pasting

```
                Cutting And Pasting
===================

   One of the main functions of any editor is to allow you to move
around chunks of text, jade makes this very easy.

   Generally, to paste down some text you have to get the text to be
inserted into the window-system's clipboard (1). If the text you wish
to paste is in one of the editor's buffers jade has a number of
commands for doing this, this is sometimes referred to as "killing" the
text.

   If the text to be pasted is in the same buffer as the position to
which you want to copy it there is an easier way than putting it into
the clipboard. For more details see
                Commands on Blocks
                 and the command
`Ctrl-i'.
```

```
   Once the text to be pasted is in the clipboard there are two
commands which can be used to insert it into the buffer before the
cursor,

`Ctrl-y'
     Inserts the contents of the standard clipboard into the buffer at
     the cursor position.

`Ctrl-Y'
     This is a variant of `Ctrl-y', it treats the string that it is
     pasting as a "rectangle" of text. That is, each successive line in
     the string (each separated by a newline character) is inserted on
     successive lines in the buffer but at the same column position.
     For more details see
                Rectangular Blocks
                 and the function
      `insert-rect'.

   ---------- Footnotes ----------

   (1)  When using an Amiga, unit zero of the `clipboard.device' is
used. For X11, the first cut-buffer.
```

1.29 jade.guide/Using Blocks

```
                Using Blocks
============

   A "block" is a section of a buffer, you mark it by specifying its
edges (ie, the first and last characters). This part of the buffer can
then have various things done to it, for example insert it somewhere
else.

   Each window can only have a single block marked at any one time, it
will be displayed in the reverse of normal text (ie white on black, not
black on white).
```

1.30 jade.guide/Marking Blocks

```
                Marking Blocks
--------------
```

To mark a block you must specify its outermost points, note that the text marked by the block ends one character before the marked position (this is so that it easy to mark whole lines).

Rectangular blocks are a bit different for more information, see

                Rectangular Blocks
                .

Note also that block marks shrink and grow as text is deleted and inserted inside them, similar to what normal marks do.

These are the commands used to mark a block,

‘Ctrl-m’
: If a block is currently marked in this window it will unmark it. Otherwise it will either mark the beginning or end of the block depending on whether or not a block has previously been partially marked.

  The normal method for marking a few characters is to first make sure that no block is currently marked (the status line displays the status of the block marks, a ‘b’ means that one end of a block has been marked and a ‘B’ means that both ends of a block are marked in which case it will be highlighted somewhere in the buffer) then press ‘Ctrl-m’ at one end, move the cursor to the opposite end and press ‘Ctrl-m’ again.

‘Meta-m’
: Set the beginning of the block to the current cursor position.

‘Meta-M’
: Set the end of the block.

‘Ctrl-x h’
: Mark the whole of the buffer.

‘ESC @’
: Mark the current word.

‘ESC h’
: Mark the current paragraph.

‘Ctrl-SPC’
: Mark from the position of the auto-mark to the cursor.

Another method for marking a block is to use the mouse, double clicking the left mouse button on a character has the same effect as moving to that character and typing ‘Ctrl-m’. Similarly, clicking the left mouse button while pressing the SHIFT key clears a marked block.

1.31 jade.guide/Commands on Blocks

```
Commands on Blocks
------------------

`Ctrl-i'
     Inserts the block marked in this window, at the cursor position,
     then unmarks the block.

`Ctrl-w'
     Copies the contents of the marked block to the standard clipboard
     and then deletes the block.

`ESC w'
     Copies the marked block to the standard clipboard, then unmarks
     the block.  This is a less destructive version of `Ctrl-w'.

`Ctrl-z'
     Deletes the text in the currently marked block.

`Ctrl-x Ctrl-l'
     Makes all alpha characters in the current block lower case.

`Ctrl-x Ctrl-u'
     Makes all characters in the block upper case.
```

1.32 jade.guide/Rectangular Blocks

```
Rectangular Blocks
------------------

   Normally blocks are thought of sequentially from their first to last
characters, this does not have to be so. It is also possible to mark
rectangles, the block marks being thought of as the opposite corners of
the rectangle.

`Ctrl-M'
     Toggle between marking sequential and rectangular blocks, each
     window has its own value for this attribute (ie one window can be
     marking rectangles while the rest don't).

`Ctrl-Y'
     Similar to `Ctrl-y' except that the string inserted is treated as a
     rectangle -- newline characters don't get inserted, instead the
     next line is inserted in the next line in the buffer at the same
     column as that inserted into the previous line. For more details
     see the function `insert-rect'.

   At present there is a problem with changing the case of a
```

rectangular block with 'Ctrl-x Ctrl-l' or 'Ctrl-x Ctrl-u', they treat it as a sequential block. This will be fixed soon.

1.33 jade.guide/Searching and Replacing

```
                Searching and Replacing
=======================
```

It is very easy to search any of jade's buffers for a specific string, the standard search command will search the current buffer for a specified regular expression.

Once you have found an occurrence of the string you are looking for it is then possible to replace it with something else.

1.34 jade.guide/Regular Expressions

```
Regular Expressions
-------------------
```

Jade uses the regexp(3) package by Henry Spencer, with some modifications that I have added. It comes with this heading:

be misrepresented as being the original software.

The syntax of a regular expression (or regexp) is as follows (this is quoted from the regexp(3) manual page):

```
A regular expression is zero or more "branches", separated by `|'.
It matches anything that matches one of the branches.

A branch is zero or more "pieces", concatenated. It matches a
match for the first, followed by a match for the second, etc.

A piece is an "atom" possibly followed by `*', `+', or `?'.  An
atom followed by `*' matches a sequence of 0 or more matches of
the atom. An atom followed by `+' matches a sequence of 1 or more
matches of the atom. An atom followed by `?' matches a match of
the atom, or the null string.

An atom is a regular expression in parentheses (matching a match
for the regular expression), a "range" (see below), `.' (matching
any single character), `^' (matching the null string at the
beginning of the input string), `$' (matching the null string at
the end of the input string), a `\' followed by a single character
(matching that character), or a single character with no other
significance (matching that character).

A "range" is a sequence of characters enclosed in `[]'. It
normally matches any single character from the sequence. If the
sequence begins with `^', it matches any single character *not*
from the rest of the sequence. If two characters in the sequence
are separated by `-', this is shorthand for the full list of ASCII
characters between them (e.g. `[0-9]' matches any decimal digit).
To include a literal `]' in the sequence, make it the first
character (following a possible `^'). To include a literal `-',
make it the first or last character.
```

Some example legal regular expressions could be:

`ab*a+b'
: Matches an `a' followed by zero or more `b' characters, followed by one or more `a' characters, followed by a `b'. For example, `aaab', `abbbab', etc...

`(one|two)_three'
: Matches `one_three' or `two_three'.

`^cmd_[0-9]+'
: Matches `cmd_' followed by one or more digits, it must start at the beginning of the line.

As well as being matched against, regexps also provide a means of "remembering" portions of the string that they match. The first 9 parenthesised expressions and the whole string that matched are recorded so that they can be used later.

The main use for this is in the command to replace a previously found regexp (`ESC p') and the Lisp functions `regexp-expand', `regexp-expand-line' and `replace-regexp'. The string which is given as

```
the template (ie, the string that replaces the matched string) is
expanded inserting these recorded strings where asked to.

   Each occurrence of `\C' in the template is a candidate for
expansion. C can be one of:

`&'
`0'
     Replaces the whole substring matched by the regular expression.

`1' to `9'
     The numbered parenthesised expression.

`\'
     The character `\'.

   For example, if a regexp of `:([0-9]+):' matches a line
`foo:123:bar', the expansion template `x_\1' would produce `x_123'.
```

1.35 jade.guide/Commands for Searching

```
Commands for Searching
----------------------

`Ctrl-s'
     Asks for a regular expression then tries to move to the start of
     the next match in this buffer.

`Ctrl-S'
     Attempts to move to the next occurrence of the regexp which was
     last entered for the `Ctrl-s' or `Ctrl-r' commands.

`Ctrl-r'
     Asks for a regexp, then moves to the start of previous occurrence
     of that regexp.

`Ctrl-R'
     Attempts to move to the previous occurrence of the regexp which
     was last entered for the `Ctrl-s' or `Ctrl-r' commands.
```

1.36 jade.guide/Commands for Replacing

```
Commands for Replacing
----------------------

`ESC p'
     Asks for a template to replace the string under the cursor (which
     should match the regexp which the search commands last looked for.
     This string is then replaced with the expansion (re the string
     under the cursor) of the template you entered.
```

```
'ESC P'
     Variant of the above, doesn't prompt for the template, uses the
     last one that you gave.
```

1.37 jade.guide/Editing Modes

```
                Editing Modes
=============

   Modes are used to tailor the editor to the *type* of the file being
edited in a buffer. They are normally a file of Lisp which installs the
buffer-local keybindings and variables which are needed for that type of
file.

   For example, C-mode is a mode used to edit C source code, its main
function is to try to indent each line to its correct position
automatically.

   At present there are only a small number of modes available. It is
fairly straightforward to write a mode for most types of files though.
```

1.38 jade.guide/Invoking a Mode

```
                Invoking a Mode
---------------

   When a new file is loaded the function 'init-mode' tries to find the
```

```
mode that it should be edited with. If it is successful the mode will be
automatically invoked. For more information see
                Automatic Mode Selection
                and the documentation for `init-mode'.
```

1.39 jade.guide/Generic-mode

```
Generic-mode
------------

   This is not a mode as such since there is no Lisp code associated
with it.  When no mode is being used to edit the buffer, it is said to
use the "Generic" mode.

   This is the base from which all other modes build, it consists of
all the standard keybindings. Words are defined as one or more
alphanumeric characters, paragraphs are separated by a single blank
line.
```

1.40 jade.guide/C-mode

```
C-mode
------

   `c-mode' is used for editing C source code files. Any files which
end in `.c' or `.h' are automatically edited in this mode.

   It's one and only function is to try and indent lines to their
correct depth, it doesn't always get it right but it works fairly well.
The keys that it rebinds to achieve this are,

`RET'
     Splits the line in two like normal. If `c-mode-auto-indent' is
     non-nil then the line that the cursor ends up on is automatically
     indented.

`Shift-RET'
     Splits the line in two, doesn't take any notice of
     `c-mode-auto-indent'.

`{'
`}'
`:'
     These keys are handled specially since the indentation of the line
     that they are inserted on may have to be adjusted.

`TAB'
     Indents the current line to what the editor thinks is the correct
     position.
```

```
'Meta-TAB'
     Moves the cursor to the next tab stop.

   Words are defined as being a sequence of alphanumeric or underscore
characters, paragraphs as being separated by a '{' as the first
character of a line.

 - Function: c-mode
     Editing mode for C source code. Automatically used for files
     ending in '.c' or '.h'.

 - Hook: c-mode-hook
     Called by 'c-mode' each time it is called.

 - Variable: c-mode-tab
     Size of tab stops used by 'c-mode'.

 - Variable: c-mode-auto-indent
     When non-nil 'RET' will indent the line after splitting it.
```

1.41 jade.guide/Lisp-mode

```
Lisp-mode
---------

   'lisp-mode' is used to edit files of Lisp intended to be read by the
editor. It is *very* basic, all it does is count the number of unmatched
parentheses in each line and indent it accordingly. I find this okay
though.

   There is also support for using a buffer as a simple shell-interface
to the editor's Lisp subsystem.

   Special keybindings are,

'RET'
     Splits the line in two like normal. If 'lisp-mode-auto-indent' is
     non-nil then the line that the cursor ends up on is automatically
     indented.

'Shift-RET'
     Splits the line in two, doesn't take any notice of
     'c-mode-auto-indent'.

'Ctrl-RET'
     Evaluates the paragraph preceding the cursor, prints the value on
     the next line.

'TAB'
     Indents the current line.

'Meta-TAB'
     Moves the cursor to the next tab stop.
```

```
'ESC Ctrl-x'
     Evaluates the paragraph before the cursor, prints it's value in
     the status line.

 - Function: lisp-mode
     Editing mode for Jade's Lisp. Automatically invoked for files
     ending in '.jl'.

 - Hook: lisp-mode-hook
     Evaluated as soon as 'lisp-mode' is called.

 - Variable: lisp-mode-tab
     Size of tabs in 'lisp-mode'.

 - Variable: lisp-mode-auto-indent
     When non-nil 'RET' indents lines after splitting them.
```

1.42 jade.guide/Texinfo-mode

```
Texinfo-mode
------------

   'texinfo-mode' is used to edit Texinfo source files, it is
automatically selected for files ending in '.texi' or '.texinfo'. It
provides a few basic keybindings to take some of the tedium out of
editing these files.

   Paragraphs are separated by the regexp '^@node', ie, each node is a
separate paragraph.

   The provided keybindings are,

'Ctrl-c Ctrl-c c'
     Insert the string '@code{}', positioning the cursor between the
     braces.

'Ctrl-c Ctrl-c d'
     Insert the string '@dfn{}', positioning the cursor between the
     braces.

'Ctrl-c Ctrl-c e'
     Inserts the string '@end'.

'Ctrl-c Ctrl-c f'
     Inserts the string '@file{}', the cursor is put between the braces.

'Ctrl-c Ctrl-c i'
     Inserts the string '@item'.

'Ctrl-c Ctrl-c l'
     Inserts the string '@lisp\n'.

'Ctrl-c Ctrl-c m'
```

```
     Inserts the string `@menu\n'.

`Ctrl-c Ctrl-c Ctrl-m'
     Prompts for the name of a node and makes a menuitem for it.

`Ctrl-c Ctrl-c n'
     Prompts for each part of a node definition (name, next, prev, up)
     and inserts the `@node ...' string needed.

`Ctrl-c Ctrl-c s'
     Inserts the string `@samp{}' and puts the cursor between the
     braces.

`Ctrl-c Ctrl-c v'
     Inserts the string `@var{}', the cursor is put between the braces.

`Ctrl-c Ctrl-c {'
     Inserts a pair of braces with the cursor between them.

`Ctrl-c Ctrl-c }'
`Ctrl-c Ctrl-c ]'
     Moves the cursor to the character after the next closing brace.

 - Function: texinfo-mode
     Mode for editing Texinfo source files.

 - Hook: texinfo-mode-hook
     Evaluated when `texinfo-mode' is invoked.
```

1.43 jade.guide/Info-mode

```
Info-mode
---------

   Despite the name of this section there is actually no such thing as
the `info-mode'. The Lisp file `info.jl' is what this section documents
-- it is a set of Lisp functions which make a buffer (the `*Info*'
buffer) into a simple browser for Info files.

   To invoke it type `Ctrl-h i' or `ESC x info', the `*Info' buffer
will be displayed showing the `(dir)' node (the root of the Info
documentation tree).

   When in the `*Info*' buffer these keybindings are in effect,

`SPC'
     Moves to the next page.

`BS'
     Moves to the previous page.

`1'
`2'
`3'
```

```
`4'
`5'
     Move to the specified menuitem (`1' means the first, etc) in the
     menu in this node.

`b'
     Move to the beginning of the current node.

`f'
     Follow a reference, the one under the cursor if it exists.  This
     command is still unimplemented.

`g'
     Prompt for the name of a node and try to display it.

`l'
     Go back to the last node that was displayed before this one.

`m'
     Prompts for a menuitem (the one on the same line as the cursor is
     the default) and display the node it points to.

`n'
     Display the next node.

`p'
     Display the previous node.

`u'
     Display the node "above" this one.

`q'
     Quit the Info browser.

   This mode has a number of disadvantages over the other Info browsers
available (ie, the standalone `info' program, or Emacs' Info viewer):

   * It depends wholly on being able to find a tag table in the Info
     file, if it can't it will complain and exit.

   * There is no support for the `*' node name.

   * As yet, no automatic following of references.

   * Seems not to work 100% with files formatted by Emacs, `makeinfo'
     formatted files work properly though.

   * No editing of modes.

 - Function: info [NODE-NAME]
     Invoke the Info viewer. If NODE-NAME is given display it, otherwise
     the node `(dir)' is used.
```

1.44 jade.guide/Using Buffers

```
                Using Buffers
=============

   As you have probably realised, buffers are probably the most
important part of the editor. Each file that is being edited must be
stored in a buffer. They are not restricted to editing files though,
all buffers are reguarded as simply being a list of lines which can be
displayed in a window and modified as needed.

   This means that they are very flexible, for example, the Lisp
debugger uses a buffer for its user interface, the Info reader uses two
buffers - one to display the current node, the other to store the
file's tag table (never displayed, just used to look up the position of
nodes).

   Each buffer has a name, generally buffers which contain proper files
use the base part of the filename, while buffers which don't correspond
to files use a word which starts and ends with asterisks (ie, `*jade*').

   Each window can display one buffer at any time. There is no
restriction on the number of windows which may display the same buffer
at once.
```

1.45 jade.guide/Displaying Buffers

```
                Displaying Buffers
------------------

   There are two main commands for switching to a different buffer,

`Ctrl-x b'
     Prompt for the name of a buffer and display it in the current
     window.

`Ctrl-x 4 b'
     In a different window (opens a new window if there is currently
     only one) prompt for the name of a buffer and display it in that
     window.
```

Both commands are very similar, the ‘Ctrl-x 4 b’ variant simply invokes a command to switch to a different window before calling the ‘Ctrl-x b’ command.

When typing the name of the new buffer you can use the prompt’s completion mechanism to expand abbreviations (see see

The Buffer Prompt

). If you just press RET with an empty prompt the default choice will be used. This will be the the buffer that was being shown in this window before the current buffer was selected (its name is displayed in the prompt’s title).

The ‘Ctrl-x Ctrl-f’ command and its variants also switch buffers since they look an existing copy of the file in a buffer before loading it from disk, see

Commands To Load Files

.

1.46 jade.guide/Deleting Buffers

Deleting Buffers

There is no real need to delete buffers, those that haven’t been used for a while just hang around at the end of the list. If you’re short on memory though it can help to kill some of the unused buffers which you have accumulated.

The command to kill a buffer is,

‘Ctrl-x k’
: Prompts for the name of a buffer (with completion) then deletes that buffer (if the buffer contains unsaved modifications you are asked if you really want to lose them). It is removed from all window’s buffer-lists and any window which is displaying it is switched to another buffer (the next in its list). Any marks which point to the buffer are made "non-resident" (that is, they point to the name of the file in the buffer) and the buffer is discarded.

1.47 jade.guide/Other Buffer Commands

Other Buffer Commands

‘ESC x rotate-buffers-forward’
: Rotates the current window’s list of buffers.

```
`ESC x recover-file'
     Loads the auto-saved copy of the file stored in this buffer
     overwriting its current contents (if any changes are to be lost
     the user will have to agree to losing them).

`ESC x revert-buffer'
     Restores the contents of the current buffer to the contents of the
     file that it was loaded from, if an auto-save file exists you are
     asked if you want to revert to that instead.

`Ctrl-x s'
     Ask whether to save any modified buffers that exist.

`ESC x clear-buffer'
     Deletes the contents of the current buffer. Beware, you *won't* be
     warned if you're about to lose any unsaved modifications!
```

1.48 jade.guide/Using Windows

```
                Using Windows
=============

   Windows have two main functions: to display the contents of buffers
(but only one buffer at a time) and to collect input from you, the user.

   The editor *must* have at least one window open at all times, when
you close the last window jade will exit, there is no limit to the
number of windows which you may have open though.

   Each window is split into two parts, they are

"The Main Display Area"
     This is the largest part of the window, it is where the buffer
     that this window is displaying is drawn.

"The Status Line"
     A single line of text associated with the window, under X11 this
     is the area of the beneath the horizontal line at the bottom of
     the window, on the Amiga it is the title of the window. The status
     line is normally used to display information about this window and
     what it is displaying, it has this format:

          BUFFER-NAME [MODE-NAME] (COL,ROW) N line(s) [FLAGS]

     Where the individual parts mean

    BUFFER-NAME
          The name of the buffer being edited, it can have either a `+'
          or a `-' appended to it, a plus means the buffer has been
          modified since it was saved, a minus means that the buffer is
          read-only.

    MODE-NAME
          Most editing modes set this to their name.
    COL
          The column that the cursor is at.

    ROW
          The row number of the cursor.

    N
          The number of lines in this buffer

    FLAGS
          General one-character flags related to the status of the
          window and its buffer.
```

Each window maintains a list of all buffers which are available for displaying, this is kept in order, from the most recently used to the least. This list (called 'buffer-list') is used by some of the buffer manipulation commands when they are working out which buffer should be displayed.

1.49 jade.guide/Creating Windows

Creating Windows

'Ctrl-x 2'
'Ctrl-x 5'
Opens a new window, it will have the most of the attributes that the current window does, things like: size, buffer, font, etc... If you are using X11 you will probably have to use your mouse to select its position, depending on the window manager you use, on the Amiga it will be created at the same position as the current window.

'Ctrl-x 4 Ctrl-f'
'Ctrl-x 4 f'
In a different window, one will be created if only one window is open, find a file, for more details see
Commands To Load Files
.

'Ctrl-x 4 a'

```
     In a different window add an entry to a change-log file. See

                Keeping ChangeLogs
                .

`Ctrl-x 4 b'
     In a different window, choose a buffer to display, similar to the
     `Ctrl-x b' command. See
                Displaying Buffers
                .

`Ctrl-x 4 h'
     Enter the help system in a different window. See
                The Help System
                .

`Ctrl-x 4 i'
     Enter the Info browser in a different window. See
                Info-mode
                .

`Ctrl-x 4 `'
     Display the next error (or whatever) in the `*compilation*' buffer
     in a different window. See
                Finding Errors
                .
```

1.50 jade.guide/Killing Windows

```
Killing Windows
---------------

`Ctrl-x 0'
     Close the current window, if it is the last window that the editor
     has open it will exit (after asking you if you wish to lose any
     unsaved modifications to buffers).

`Ctrl-x 1'
     Close all windows except the current one.
```

1.51 jade.guide/Other Window Commands

```
Other Window Commands
---------------------

`Ctrl-x o'
     Activate the next window of the editor's. Under X11 this involves
     warping the [mouse-]cursor to the top left corner of the newly
     activated window.
```

1.52 jade.guide/Using the Prompt

```
                Using the Prompt
================

   There are two different styles of prompt that the editor uses when it
wants you to enter a string.



                 The Simple Prompt
                    The prompt at the bottom of the window

                 The Buffer Prompt
                    Prompt with its own buffer and completion
```

1.53 jade.guide/The Simple Prompt

```
The Simple Prompt
-----------------

   The simplest prompt uses the the bottom-most line in the window, it
prints the prompt’s title on the left hand side, you should type your
response and then press the RET key. This prompt is very primitive, the
only special commands that it has are,

‘BS’
     Delete the previous character.

‘UP’
‘DOWN’
     Replace the contents of the prompt with the last string entered.
     When you type ‘UP’ or ‘DOWN’ again the original contents are
     restored.

‘ESC’
     Cancel the prompt.

All other keys are simply printed in the prompt -- whatever they are.
```

1.54 jade.guide/The Buffer Prompt

```
The Buffer Prompt
-----------------
```

This type of prompt is more sophisticated. It creates a new buffer for you to type your response into (called `*prompt*'), the title of the prompt is displayed in the buffer's first line.

Normally you type the answer to the prompt into the buffer and then press the RET key. All normal editor commands are available while you are using the prompt, you can switch buffers, load new files, whatever you like.

Another advantage of this type of prompt is that it supports "completion", this allows you to type the beginning of your response then press the TAB key. What you have typed will be matched against the list of responses that the editor has (ie, when being prompted for the name of a file it will be matched against all available files), if a unique match is found your response will be completed to that match.

If several potential completions are found, these will be displayed after the line `::Completions::' in the buffer and your response will only be completed as far as the potential completions are similar. For example, if you enter `fo' then press TAB and files called `foo' and `foobar' exist, the contents of the prompt will become `foo'.

Completion is provided for many different things, some are: files, buffers, symbols, functions, variables, Info nodes, etc...

The special commands for this type of prompt are,

`TAB'
    Complete the contents of the prompt. If more than one potential completion exists they are printed in the buffer.

`RET'
    Enter the result of this prompt. If you press RET while the cursor is on a printed potential completion (those under the `::Completions::' line) the whole line will be entered. Otherwise, just the text to the left of the cursor is entered.

`ESC ?'
    Print all possible completions of the current prompt but do not try to actually change the contents of the prompt.

`Ctrl-g'
    Cancel the prompt.

1.55 jade.guide/Using Marks

Using Marks
===========

Marks are used to record a position in a file, as the file's buffer is modifed so does the position that the mark points to -- a mark will keep pointing at the same character no matter what happens (unless the character is deleted!).

The other good thing about marks is that they point to files *not* buffers. This means that you can set a mark in a buffer, delete the buffer and then move to the position of the mark, the file will be reloaded and the cursor will point at the original character.

Normally there are three user-accessible marks (1) and one special 'auto-mark' which is used, amongst other things, to record the "previous" position of the cursor, allowing you to retrace your last major step.

The commands available on marks are,

'F1'
'F2'
'F3'
: Move to the mark #1, #2 or #3, depending on which function key is pressed (F1 means mark #1, etc...). If the file pointed to is not in memory it will be loaded into a new buffer.

'Shift-F1'
'Shift-F2'
'Shift-F3'
: Set the position of mark #1, #2 or #3, depending on the function key.

'Ctrl-x Ctrl-x'
: Swap the positions of the cursor and the 'auto-mark'.

'Ctrl-@'
: Set the position of the 'auto-mark'.

---------- Footnotes ----------

(1) There is no reason why you can't have more, the editor sets no limitation on the number of marks available. This is just how I have set the editor up.

1.56 jade.guide/Compiling Programs

Compiling Programs
==================

Jade has a number of features to help you develop programs, foremost is the ability to run a compilation inside one of the editor's buffers. Unfortunately, this is only possible when using the Unix operating system at the present.

Once the compilation has finished you can then step through each error produced.

Running a Compilation

1.57 jade.guide/Running a Compilation

```
Running a Compilation
---------------------

   The command to run a shell command in a buffer is,

`ESC x compile'
     Prompts you for the command to execute, with a default of the last
     command you ran (starts as `make'). A shell process is created
     which runs asynchronously to the editor in the same directory as
     the current buffer's file was loaded from. The buffer
     `*compilation*' is selected and this is where all output from the
     program is printed.

   When the process finishes running a message is printed in the
`*compilation*' buffer telling you its exit-code.

   Only one process may be run with the `compile' function at once.

   This command is not available on the Amiga version yet.
```

1.58 jade.guide/Finding Errors

```
Finding Errors
--------------

   When you have compiled something with the `ESC x compile' command it
is possible to step through each of the errors that it produces. To do
this use the command,

`Ctrl-x `'
`ESC x next-error'
     Displays the next error in the `*compilation*' buffer. The file
     that is in is loaded (if necessary) and the line with the error is
     found.

   If you edit a file which has errors in it, then try to find the next
error (which is in the same file) everything will still work. The
positions of errors are updated as the buffers are modified.
```

The only exception to this is when you invoke the 'next-error' function while the '*compilation*' buffer is still being written to. If more errors are produced in a file which has been modified since the compilation started it is likely that the positions will get out of sync.

By default, the 'next-error' function understands the type of error output that 'gcc' produces. This is of the form,

```
FILE:LINE-NUMBER:DESCRIPTION
```

It is possible to use other formats though, the variables which control this are,

- Variable: compile-error-regexp
  Regular expression to match a line containing an error. For 'gcc' this is '^(.*):([0-9]+):(.+)'.

- Variable: compile-file-expand
  Expansion template to produce the name of the file with the error, using 'compile-error-regexp' and the line containing the error. By default this is '\1'.

- Variable: compile-line-expand
  Similar to 'compile-file-expand' except that it expands to a string defining the number of the line with the error. By default, '\2'.

- Variable: compile-error-expand
  Similar to 'compile-file-expand', but produces the description of the error. By default, '\3'.

1.59 jade.guide/Using Grep

```
                Using Grep
----------
```

It is often very useful to grep through a set of files looking for a regular expression, this is what the 'grep' command does. With jade it is possible to run an external 'grep' program in the '*compilation*' buffer. This then enables you to step through each grep hit using the 'Ctrl-x '' command,
Finding Errors
.

The commands to use grep are,

'ESC x grep'
: Prompt for a string of arguments to give 'grep', you do not need to provide the name of the program, or the '-n' switch, this is done automatically. The shell will do any filename-globbing on the arguments so it is advisable to surround the regular expression with single quotes.

     Note that the regular expression syntax will be different to that
     which jade uses. Also this command won't work on an Amiga.

'ESC x grep-buffer'
     This command provides a method for scanning the current buffer for
     all lines matching a regular expression (which you are prompted
     for). It is written entirely in Lisp -- this means that the normal
     regular expression syntax is needed and it will work on an Amiga.

1.60 jade.guide/Keeping ChangeLogs

Keeping ChangeLogs

A ChangeLog is a file (usually called 'ChangeLog') which keeps a log of all changes you have made to the files in its directory. For example, the 'src/ChangeLog' file for Jade keeps a list of changes made to the editor's source code.

There is no magic involved, you simply use a command to add a new entry to a directory's log after modifying a file in that directory. You then have to enter a summary of the changes that you made.

The command to do this is,

'ESC a'
     Prompts for the name of a directory then lets you type a
     description of the changes you have made.

If you enter more than one change in the same day (and from the same host) the same heading will be used. The heading consists of the time and date, your name, your login and the name of the host you're on. (1)

---------- Footnotes ----------

(1) On the Amiga there is no way to get these details. So, Jade looks for some environment variables, 'USERNAME' for the login name, 'HOSTNAME' for the name of the host and 'REALNAME' for your actual name.

1.61 jade.guide/Simple Customisation

Simple Customisation
====================

The best way to tailor the editor to your own requirements is with your personal startup file. This is called '.jaderc' in your home directory (1), it is a file of Lisp forms evaluated when Jade initialises itself.

Usually, setting the values of variables in your startup file is enough to configure Jade how you want, the Lisp function to set a variable is called ‘setq’, it’s first argument is the name of the variable, it’s second the value you wish to set it to. Normally this value will be one of the following data types,

‘"xyz"’
    A string ‘xyz’.

‘123’
‘0173’
‘0x7b’
    A number, all of the above have the value 123 (in decimal, octal and hexadecimal).

‘nil’
‘t’
    A boolean value, ‘nil’ means false, or not true. ‘t’ is the opposite (in fact, any value not ‘nil’ is true).

My ‘.jaderc’ file looks something like this (note that semicolons introduce comments),

```
;; Size of tabs for C source is 4
(setq c-mode-tab 4)

;; Size of tabs for Lisp source is 2
(setq lisp-mode-tab 2)

;; On X11 scroll quarter of a screen at once, else a line at a time
(setq y-scroll-step-ratio (if (x11-p) 4 0))

;; When on an Amiga, flag that I don’t want pulldown menus
(when (amiga-p)
  (setq amiga-no-menus t))
```

---------- Footnotes ----------

(1) On the Amiga, your home directory is defined as the contents of the environment variable ‘HOME’.

1.62 jade.guide/Programming Jade

Programming Jade
****

Unfortunately I haven’t written this section yet. If you wan’t to program Jade your best bet is to look at the files in the ‘lisp/’ directory. Online documentation is available for all editor functions,

The Help System
.

If you don’t know Lisp look at any Lisp book. Jade’s Lisp is fairly

```
similar to Emacs-Lisp (though the editor-related functions differ
greatly) so a good starting point may be the Emacs-Lisp manual.
```

1.63 jade.guide/Reporting Bugs

```
Reporting Bugs
**************

   If you think you've found a bug in Jade I want to know about it,
there is a list of problems that I am aware of in the `BUGS' file, if
your's appears in here tell me anyway to make me fix it.

   When submitting bug reports I need to know as much as possible, both
about the problem and the circumstamces in which it occurs. In general,
send me as much information as possible, even if you think it's probably
irrelevant.

   If you can, contact me via email, my address is `jsh@ukc.ac.uk'.  If
you don't get a reply within about a week it's probably a university
vacation -- this means that I won't get your message for a while, if
it's important try my postal address, this is,

     John Harper
     91 Springdale Road
     Broadstone
     Dorset
     BH18 9BW
     England

   As well as bugs I'm interested in any comments you have about the
editor, even if you just tell me you hate it (as long as you say *why*
you hate it!).
```

1.64 jade.guide/Function Index

```
                Function Index
**************
```

1.65 jade.guide/Variable Index

```
               Variable Index
**************
```

1.66 jade.guide/Key Index

Key Index
****

1.67 jade.guide/Concept Index

Concept Index
****