Jed 0.99.18 Quick Reference, v. 1.0.9

Guido Gonzato and John. E. Davis

March 18, 2010

Contents

1 Introduction **3**
1.1 Getting Started . . . 3
1.2 Getting Help . . . 5
1.3 Notes for Windows Users . . . 5
1.4 Startup Files . . . 6
1.5 Customisation . . . 6
1.6 Emulations . . . 7
1.7 Window Size, Font, and Colour . . . 7
1.8 Extending the Jed Library . . . 8
1.9 Mini-Buffer . . . 8
1.9.1 Command Line Editing and Completion . . . 8
1.9.2 File Names . . . 9
1.10 Buffers, Windows, and Status Line . . . 9
1.11 Backup and Auto-Save Files . . . 10
1.12 Batch and Script Mode . . . 11

2 Basic Editing **11**

3 Main Features **13**

4 Editing **13**
4.1 Menus . . . 13
4.2 Marking Text . . . 14
4.3 Using the Mouse . . . 14
4.4 Pasting from Other Applications . . . 14
4.5 Reformatting Paragraphs . . . 15
4.6 Keyboard Macros . . . 16
4.7 Repeating Actions . . . 16
4.8 Rectangles . . . 16
4.9 Block Sorting . . . 17

4.10 8-bit Characters and Mutekey . . . 17
4.11 Abbreviations and Completion . . . 17
4.12 Ispell . . . 18
4.13 Registers . . . 18
4.14 Tab Issues . . . 18
4.15 Regular Expressions . . . 18
4.16 Occur Search . . . 19

5 Buffer-related Issues 19
5.1 Window Splitting . . . 19
5.2 Folding . . . 19
5.3 Narrowing Regions . . . 20
5.4 Compiling . . . 20
5.5 Tags . . . 20
5.6 Commenting Code . . . 20
5.7 Buffer Editor . . . 20
5.8 Mail/Rmail (Unix Only) . . . 20
5.9 Man and Info Reader . . . 21
5.10 Quick Tips . . . 21

6 File-Related Issues 22
6.1 Recovering Files . . . 22
6.2 Editing Compressed Files . . . 22
6.3 File Locking (Unix Only) . . . 22
6.4 End-Of-Line Issues . . . 22
6.5 Multiple Versions . . . 22
6.6 RCS and SCCS interface . . . 23
6.7 Directory Editor . . . 23

7 Modes 23
7.1 Byte-Compiling Modes . . . 24
7.2 Syntax Highlighting . . . 24
7.3 C mode . . . 25
7.4 Fortran Mode . . . 25
7.5 LaTeX Mode . . . 26
7.6 Tex Mode . . . 27
7.7 HTML Mode . . . 27
7.8 Docbook SGML Mode . . . 28
7.9 Most Mode . . . 28

8 Advanced Customisation 28

8.1 UTF8 Support 28
8.2 Setting Keybindings 29
8.3 Loading Additional .sl Files 29
8.4 Autoloading Functions 29
8.5 Hooks 30

9 The End **30**

1 Introduction

From the JED home page:

"JED is a freely available text editor for UNIX, VMS, MSDOS, OS/2, BeOS, QNX, and win9X/NT platforms. Although it is a powerful editor designed for use by programmers, its drop-down menu facility makes it one of the friendliest text editors around. Hence, it is ideal for composing simple email messages as well as editing complex programs in a variety of computer languages.

JED makes extensive use of the S-Lang library, which endows it with a powerful S-Lang macro language."

S-Lang files implement Modes (§ 7), i.e. specialised editing facilities. In addition to the modes included in the distribution, many more are available. And yes, you can write your own.

JED was written by John E. Davis, `davis@space.mit.edu`. Some useful addresses:

- the JED home page, http://www.jedsoft.org/jed
- the JED users mailing list, http://www.jedsoft.org/jed/mailinglists.html
- the JED modes repository, http://jedmodes.sourceforge.net

This manual assumes that JED (version 0.99.18) is properly installed on your system. For installation details, please see the `INSTALL*` files included in the distribution.

As the author of JED states, the version number is still below 1.0 *"because of the lack of adequate documentation"*. This quick reference is hopefully a good starting point, but don't forget to consult the `.txt` files in the distribution for further explanations.

1.1 Getting Started

Depending on the operating system, the name of the JED executable can be `jed` (console), `jed386` (DOS), `xjed` (X11), or `wjed` (Windows).

```
prompt> jed [switches] [file(s)]
```

The following command-line switches are available:

```
Switch          Meaning
-----------------------
-batch          run JED in batch (non interactive) mode.
-help           shows usage
```

```
-hook fn (argv)  exec fn with rest of argv parameters
-script          like '-batch -l', but no output
-tmp             do not backup buffer
--ANYTHING       execute ANYTHING as a function
-e <emulation>   use emulation (e.g. emacs, ide, cua...)
-n               do not load jed.rc (.jedrc) file.
-g <n>           goto line 'n' in buffer
-l <file>        load 'file' as S-Lang code.
-f <function>    execute S-Lang function named 'function'
-s <string>      search forward for 'string'
-2               split window
-i <file>        insert <file> into current buffer
```

The directory defined in the `JED_ROOT` environment variable contains the installation. Documentation files reside in `JED_ROOT/doc`, S-Lang files in `JED_ROOT/lib`, info pages in `JED_ROOT/info`, and binaries in `JED_ROOT/bin`. Binaries may be moved to a standard system directory.

Common values for `JED_ROOT` in UNIX environments are `/usr/local/jed` or `/usr/share/jed`.

`xjed` accepts additional switches:

```
-Display       [-d]     display to run on
-Name                   instance name
-Geometry               initial geometry specifications
-font          [-fn]    default font to use
-background    [-bg]    background color
-foreground    [-fg]    foreground color
-Title                  name to be displayed on the title bar
-fgStatus      [-sfg]   foreground color of the xjed buffer status line
-bgStatus      [-sbg]   background color of the xjed buffer status line
-fgRegion      [-rfg]   foreground color of a region as defined by point
                        and mark
-bgRegion      [-rbg]   background color of a region as defined by point
                        and mark
-fgCursor      [-cfg]   text cursor foreground color
-bgCursor      [-cbg]   text cursor background color
-fgMouse       [-mfg]   mouse cursor foreground color
-bgMouse       [-mbg]   mouse cursor background color
-fgMenu        [-fgm]   foreground color for menubar
-bgMenu        [-bgm]   foreground color for menubar
-fgMessage     [-fgms]  foreground color for messages
-bgMessage     [-bgms]  background color for messages
-fgError       [-fger]  foreground color for error messages
-bgError       [-bger]  background color for messages
-fgOperator    [-fgop]  foreground color for operators (+, -, etc...)
-bgOperator    [-bgop]  background color for operators
-fgNumber      [-fgnm]  foreground color for numbers
-bgNumber      [-bgnm]  background color for numbers
-fgString      [-fgst]  foreground color for strings
-bgString      [-bgst]  background color for strings
-fgComments    [-fgco]  foreground color for comments
-bgComments    [-bgco]  background color for comments
-fgKeyword     [-fgkw]  foreground color for keywords
-bgKeyword     [-bgkw]  background color for keywords
-fgKeyword1    [-fgkw1] foreground color for keywords1
```

```
-bgKeyword1    [-bgkw1] background color for keywords1
-fgKeyword2    [-fgkw2] foreground color for keywords2
-bgKeyword2    [-bgkw2] background color for keywords2
-fgDelimiter   [-fgde]  foreground color for delimeters
-bgDelimiter   [-bgde]  background color for delimeters
-fgPreprocess  [-fgpr]  foreground color for preprocessor lines
-bgPreprocess  [-bgpr]  background color for preprocessor lines
```

For example,

```
xjed -d space:0.0 -font 9x15 -bg blue -fg white
```

will start `xjed` using the server called `space`, using a white on blue 9x15 font.

You can set defaults for `xjed` inserting appropriate lines in your `.Xdefaults`, e.g.

```
xjed*Geometry: 80x40+100+100
xjed*font: 9x15bold
```

(If `.Xdefaults` is missing, just create it.)

If you're using `xjed` with UTF8 support (see UTF8 Support (§ 8.1)), line drawings around menus might show as strange characters. In that case, add this line to `.Xdefaults`:

```
UXjed*font: -misc-fixed-medium-r-normal--15-140-75-75-c-90-iso10646-1
```

To reload the X settings, type:

```
xrdb -load ~/.Xdefaults
```

1.2 Getting Help

There are several ways to get help:

- via the *Help* menu: there are many options, try them all;
- typing `Esc X help`, you access the help screen pertinent to the emulation you're using;
- browse the off-line documentation that JED ships with;
- look at the `*.sl` sources in the `JED_ROOT/lib` directory.

1.3 Notes for Windows Users

First of all: on Windows NT/2000/XP etc., JED must be installed by the system administrator. A standard installer is provided, but it does not extend the PATH to make the console `jed` executable readily available.

If you don't want to or don't know how to modify the environment to make JED work, use the following `.bat` file. In this example, we'll assume that JED is installed in `C:\TOOLS\JED`:

```
rem wjed.bat
rem
SET PATH=C:\TOOLS\JED\BIN;%PATH%
SET JED_ROOT=C:\JED\TOOLS
C:\TOOLS\JED\BIN\JED.EXE %1 %2 %3 %4 %5 %6 %7 %8 %9
```

Make a shortcut to this `.bat` file in `C:\WINDOWS\SENDTO`. By right-clicking on a file, you will be able to select "Wjed" from the "Send to..." menu.

`wjed` is a bit confusing when it comes to cut, copy, and paste. The *Edit* menu cannot be used to paste text; only the *right* mouse button works.

Console JED for DOS/Windows may not insert characters like '#', '@' and in general those characters that are obtained via the `Alt Gr` key on a non-US keyboard. This can be fixed inserting these lines in `.jedrc` (alas, using another editor):

```
setkey ("self_insert_cmd", "\e@");
setkey ("self_insert_cmd", "\e#");
setkey ("self_insert_cmd", "\e[");
setkey ("self_insert_cmd", "\e]"); % and so on...
```

Currently, `wjed` doesn't honour the standard menus. To get the same menus as console JED, you will have to edit the system file `os.sl` to uncomment the line that reads:

```
% . "menus" evalfile pop         %  Uncomment to enable text menus
```

1.4 Startup Files

When JED starts up, it loads an initialization file, `site.sl`; then, if they exist, `defaults.sl` and `/etc/jed.conf`; then it tries to load your personal initialization file, `$HOME/.jedrc`; failing that, it will load a system-wide file, `jed.rc`. All of these files are written in S-Lang.

`site.sl` is particularly important. In addition to defining keybindings, functions and variables, it also declares *autoloads*. These are associations between functions and the `.sl` file where they are defined. When such a function is needed, the corresponding `.sl` file will be loaded. This arrangement allows JED to load only needed functions, saving memory and startup time.

System administrators will want to provide a system-wide `defaults.sl`. It's desirable that editor emulation (see below) not be included in system-wide init files, so that users can use the `-n -e` switches. More details in Section Batch and Script Mode (§ 1.12).

1.5 Customisation

You don't have to be proficient in S-Lang to tailor JED to your needs. On UNIX, copy `jed.rc` to `$HOME/.jedrc`, read it carefully to become familiar with the features you can customise, then make changes. On Windows, modify `jed.rc` directly, but make a backup copy beforehand.

In many cases, all you have to do is change the value of JED's global variables.

Apart from modifying existing parameters, sooner or later you will want to modify your `.jedrc` heavily, perhaps adding your own S-Lang functions. You could simply append lines to the end, but the best place is often within the braces that define the block:

```
if (BATCH == 0)
{
...
}
```

Everything in the block will not be loaded when JED is used in batch mode, making it faster to load. For further details, please read Advanced Customisation (§ 8).

1.6 Emulations

The very first change you may want to make is choose your favourite editor emulation. `emacs` emulation is the default; DOS/Windows users might like CUA mode, which uses standard Windows keybindings, or IDE mode, which is compatible with Wordstar and the Borland Turbo Pascal/C++ editors.

Edit your `.jedrc` and uncomment the appropriate line removing the leading `%`:

```
%  () = evalfile("emacs");        % Emacs-like bindings
%  () = evalfile("edt");          % EDT emulation
%  () = evalfile ("ide");         % Borland IDE (see also doc/ide-mode.txt)
%  () = evalfile ("brief");       % Brief Keybindings (MSDOS only!!)
%  () = evalfile("wordstar");     % obsolete --- use ide instead)
%  () = evalfile ("cua");         % CUA-like key bindings
```

Every emulation reserves a key for extended operations. In `emacs` mode, this key is `Ctrl-C`; in IDE mode, `Ctrl-Z`; in CUA mode, `Ctrl-E`. You can change the key inserting a line like this in `.jedrc`:

```
_Reserved_Key_Prefix = "^Y";
```

From now on, we'll assume that JED is running in Emacs mode. To find out the key bindings associated to a function in other emulations, select *Help/Where Is Command* and type the function name. For instance, querying JED in IDE mode about the `comment_region` function, you will receive the answer `Ctrl-K ;`.

In the following, I will not provide key bindings but functions names whenever possible. It's up to you to find the associated key binding, if any.

1.7 Window Size, Font, and Colour

To change font size and dimensions in `xjed`, edit the file `.Xdefaults` in your home directory and add these two lines:

```
xjed*Geometry: 80x35+150+0 ! columns x lines, X and Y offset
xjed*font: 7x14
! or:
! xjed*font: lucidasanstypewriter-14
```

Other valid values for font size are 5x7 (tiny), 6x10, 6x12, 6x13, 7x13, 7x14, 8x13, 8x16, 9x15, 9x18, 10x20, 12x24, and 18x18 (huge). Use `xlsfonts | less` to list available fonts.

To do the same in `wjed`, open `C:\WINDOWS\WIN.INI` and search for the `[WJED]` section. If it's missing, create it. You will modify the properties with lines like these:

```
X=67
Y=14
Width=696
Height=542
Font=Fixedsys
FontHeight=10
```

There are several colour schemes available; you will find them in the *Windows/Color Schemes* menu.

1.8 Extending the Jed Library

You may want to install additional `.sl` files, but normally you can't add them to `JED_ROOT/lib`; only the system administrator can do that. In this case, you can make a directory where you will keep your own `.sl` files. You will also have to add a few lines to `$HOME/.jedrc`.

The settings shown below force JED to look for `.sl` files in `$HOME/mylib/jed` before the standard location. The same holds for highlighting patterns and colour schemes.

```
variable Jed_Home_Library = "~/mylib/jed";
set_jed_library_path (Jed_Home_Library + "," + get_jed_library_path ());
Jed_Highlight_Cache_Dir = Jed_Home_Library;
Jed_Highlight_Cache_Path = Jed_Highlight_Cache_Dir + "," +
                           Jed_Highlight_Cache_Path;
Color_Scheme_Path = Jed_Home_Library + "/colors/," + Color_Scheme_Path;
```

1.9 Mini-Buffer

The *Mini-Buffer* consists of a single line located at the bottom of the screen; much of the dialog between the user and JED takes place in this buffer. For example, when you search for a string, JED will prompt you for the string in the mini-buffer.

You can type commands in the mini-buffer pressing `Esc-x` (or `Alt-x`); the `M-X` prompt will appear. At this stage, insert a function name. For example, to move to the top of a buffer you will type `Alt-X bob`.

The mini-buffer also provides a direct link to the S-Lang interpreter. To access the interpreter, press `Ctrl-X Esc` (or `Esc X` in IDE mode) and the `S-Lang>` prompt will appear in the mini-buffer. Enter any valid S-Lang expression for evaluation by the interpreter.

In all cases, you can quit the mini-buffer pressing `Ctrl-G`.

1.9.1 Command Line Editing and Completion

It is possible to recall data previously entered into the mini-buffer by using the Up and Down arrow keys. This makes it possible to use and edit previous expressions in a convenient and efficient manner.

Commands typed in the mini-buffer can be completed pressing the `Tab` and/or `Space` keys. This is useful when you don't remember the name of a function (JED provides hundreds of them), or to cycle amongst file or buffer names.

For example, let's suppose you want to see all functions that start in 'b'. Type `Alt-X b`, then press `Tab`: a new window will appear containing a list of possible completions. Now press `Space` repeatedly and note how the command line completes itself.

Not all functions made available by JED support completion. For instance, functions defined in emulation modes won't self complete; functions defined in files not yet loaded will not complete either. More details in Section Autoloading Functions (§ 8.4).

1.9.2 File Names

When JED prompts you for a file name, completion works as explained above. The file name syntax is operating system-dependent. Valid examples are:

```
C:\JED\LIB\JED.RC
~/progs/jed/ide.sl
dev$user:[davis.jed]vms.c
```

which denote file paths for DOS/Windows, UNIX, and VMS. In UNIX, the tilde `~` expands to the user's `HOME` directory.

Wildcards are allowed. For example, to select one of the files with extension `.txt` in the `text` subdirectory, you can type

```
text/*.txt
```

in the mini-buffer, then press `Tab` and `Space` as usual.

1.10 Buffers, Windows, and Status Line

JED supports multiple windows. Each window may contain the same buffer or different buffers. Buffers are usually (but not necessarily) attached to a file.

A default buffer called `*scratch*` exists that can be used to scribble things that need not be saved.

A status line is displayed at the bottom of each window. The status line contains information such as the JED version number, the buffer name, the mode, etc. On the left-hand side, the status bar starts with a string of dashes. In this area, special *indicators* can appear:

`**` buffer has been modified since last save.

`%%` buffer is read only.

`m` mark set indicator; this means a region is being defined.

`d` file changed on disk indicator. This indicates that the file associated with the buffer is newer than the buffer itself.

`s` spot pushed indicator.

`+` Undo is enabled for the buffer.

`[Macro]` a macro is being defined.

`[Narrow]` buffer is narrowed to a region of lines. Please see Section Narrowing Regions (§ 5.3).

To customise the status line, you may use this command in your `.jedrc`:

```
set_status_line (" Jed %v: %b (%m%n) (%l %c) %t", 1);
```

The meaning of the fields is:

`%b` buffer name

`%f` file name (without the directory part)

`%F` file name with directory

`%v` JED version. If followed by `U`, means UTF8 active

`%t` current time — only used if variable DISPLAY_TIME is non-zero

`%p` line number or percent string. If LINENUMBERS is 2, this expands to "line number,column number"

`%c` column number

`%%` literal '%' character

`%m` mode string

`%a` If abbrev mode, expands to "abbrev"

`%n` If buffer is narrowed, expands to "Narrow"

`%o` If overwrite mode, expands to "Ovwrt"

`%O` Overwrite/Insert flag - like %o, but shows INS/OVR

`%l` Shows current line number

`%L` Shows number of lines in the file

1.11 Backup and Auto-Save Files

By default, JED creates backup files appending `~` to the file name. This feature can be disabled setting the `No_Backups = 1` in `.jedrc`.

Moreover, JED periodically auto-saves its buffers; auto-save files are prefixed with `#`. The auto-save interval may be changed by setting the variable `MAX_HITS` to the desired value; the default is 300 "hits" (key presses) on the buffer.

More details in Section Recovering Files (§ 6.1).

1.12 Batch and Script Mode

JED can be used to perform text editing tasks non interactively. For example,

```
jed -batch -l preparse
```

starts JED in batch mode, then loads and runs the file `preparse.sl`. This will generate some verbose output.

To use JED as a non-interactive text processing facility, you will want to use the `-script` option followed by the name of an S-Lang file. All other switches after `-script` are ignored, and the user's `.jedrc` is not loaded.

Let's see a simple example. Save the following code as `hello.sl`:

```
% a very simple S-Lang program
flush ("Hello, world!");
```

Type `jed -script hello.sl`: as you may expect, `Hello, world!` will be displayed. Obviously, real S-Lang programs will prove more useful. Scripting makes JED a valuable alternative to other tools like `sed`, which are often cryptic to use.

On UNIX systems, you can write shell scripts that use JED as command interpreter. For example, save this script as `replace.sh`:

```
#!/usr/bin/jed -script

% this script replaces text in a file.
% __argv[0] = /usr/bin/jed
% __argv[1] = -script
% __argv[2] = ./change.sh
% __argv[3] = file.txt
% __argv[4] = old
% __argv[5] = new

if (__argc != 6) {
  message ("Usage: ./replace.sh filename old-string new-string");
  quit_jed ();
}

() = read_file (__argv[3]);
replace (__argv[4], __argv[5]);
save_buffer ();
quit_jed ();
```

2 Basic Editing

`M-Key` stands for `Esc-Key` or, equivalently, `Alt-Key`; `^Key` stands for `Ctrl-Key`. These are the very basic commands to get you started:

Operation	Emacs	IDE	CUA
quit jed	^X^C	^Kx	M-Q
quit the cmd line	^G	^G	^G
suspend	^Z	^Kz	?
	File Operations		
find (open) file	^X^F	^Ke or F3	^O
insert file	^Xi	^Ki	M-I
save	^X^S	^Kd or F2	^S
save as	^X^W	^Ks	n/a
close file	^Xk	^Kq or M-F3	n/a
change buffer	^Xb	^Kp or F6	n/a
	Movement		
word left	M-b	^A	M-Left
word right	M-f	^F	M-Right
beginning of line	^A	^Qs	^A
end of line	^E	^Qd	^E
page up	M-c	^R	^P
page down	^V	^C	^N
top of buffer	M-<	^Qr	M-^A
end of buffer	M->	^Qc	M-^E
line no.	M-X goto_line_cmd	^Qi	M-X goto_line_cmd
	Deleting		
character left	^H or BS	^H or BS	^H or Bs
character right	^D	M-g	^D
word left	M-DEL	M-BS	^-Bs
word right	M-d	^T	^-Del
line	^A^K	^Y	^K
to end of line	^K	^Qy	M-K
undo	^Xu	^U	^Z
redo	^G^Xu	^G^U	^G^Z
	Search and Replace		
search	^S	^Qf	^F
replace	M-%	^Qa	^R
repeat search	n/a	^L	n/a
	Regions (aka Blocks)		
begin	^@ or ^-SPACE	^Kb	Shift-arrow
cut	^W	^Ky	^X
copy	M-W	^Kh	^C
paste	^Y	^Kc	^V

Moreover, all emulations support these additional key bindings:

^_	undo
M-q	format_paragraph
M-n	narrow_paragraph
M-\	trim_whitespace
M-!	do_shell_cmd
M-$	ispell
M-u	upcase_word
M-c	capitalize_word
M-.	find_tag

You should also be aware of these important features:

- JED acts in different ways, according to the type of text you're writing; that is, it supports Modes (§ 7). Some characters are not always typed in as you would expect, and some key bindings may stop working.
- The inverted quote character ‘ (ASCII 96) is used to insert the next character literally. For example, ‘ Esc will insert a real Esc character (ASCII 27) in the text. To get an inverted quote, type ‘ ‘.
- To insert a character of a given ASCII code, press Esc, then the character ASCII code, then ‘. For example, to insert an open curly brace you can type Esc 123 ‘.
- These are "normal double quotes", these are "smart quotes". If you get smart quotes but want normal double quotes, type ‘ then ". The same applies to single quotes.

3 Main Features

So many things to list here! Many of these features are accessible via menus, others via key bindings, others still via the mini-buffer typing M-X followed by the name of the relevant function. For some of them, a brief explanation is provided below. There you are, in no particular order:

> pull-down menus; mouse support; multiple windows/buffers; mini-buffer completion and history; directory/buffer manager; 'modes' for editing program sources or markup languages like LaTeX or HTML; recovery of lost files; file locking; loading of compressed files; folding support; 8-bit characters and mute-key support; abbreviations and completion; shell and compiler interface; numbered backups; RCS support; bookmarks; calendar; matching parenthesis highlighting/moving to; filtering of regions of text; editing of binary files; info and man page reader; incremental search; ispell word or automatic on the whole buffer; macros; mail and rmail; 'occur' search; plain and regular expression search and replace; replace across several buffers; registers; block sorting; rectangular cut and paste; editable tab positions; tab-space conversion; Windows-like cut and paste; reformatting of paragraphs; CR/LF independence; key codes; VMS help; mime filtering.

In the following sections, only the names of the functions will be reported; find out the key bindings, if any, via *Help/Where Is Command*. Look at the S-Lang files if you're curious.

4 Editing

4.1 Menus

Pull-down menus are supported in all terminals, and are accessed via the F10, arrows and Enter keys; or directly via M-key or the mouse. If Alt-key doesn't work, try Esc-key.

You will want to append a list of recently accessed files to the *File* menu. Just insert this line in .jedrc:

```
() = evalfile ("recent.sl");
```

You can roll your own menus by customizing popups.sl.

4.2 Marking Text

Regions (blocks) of text are marked with `set_mark_command`, or emulation-dependent equivalent commands; please see the table in Section Basic Editing (§ 2). A region is defined by two screen locations called *point* and *mark*, which respectively start and end the region.

Many commands operate on regions, i.e. for deleting, filtering, conversion, and so on. For instance, `xform_region ('u')` will turn the region to upper case.

4.3 Using the Mouse

Mouse support is available for DOS/Windows, Linux console, Xjed, and JED in several favours of xterms:

- Click the *left* mouse button to move the cursor to another location.
- To copy a region for subsequent pasting, press the *left* mouse button and drag the mouse to the end of the region.
- To paste a previously copied region, press the appropriate button:
  - *right* (Linux console and `wjed`)
  - *middle* (`xjed`)
  - `Alt`-*middle* (DOS console)
- To cut a region and put it in the paste buffer, define a region by dragging with the *right* (console, `xjed`) mouse button. Now release the *right* button and press it again.
- To paste from another window/application into Jed, press the *Alt* key while pressing the *right* (Linux console) button.

If you're using JED in `rxvt`, `xterm` or other terminal emulator, the above operations are obtained this way:

- To define a region, click `Shift`-*left*;
- To paste from the pastebuffer, move the cursor to where you want to paste and press the `Shift`-*middle*.

On Windows, you can paste text from other applications pressing the *right* button.

S-Lang files: `mouse.sl`, `mousex.sl`, `mswmouse.sl`

4.4 Pasting from Other Applications

Pasting may produce unexpected indentation. Let's suppose you select the following code snippet in a web browser:

```
untgz()
{
  if [ $# != 0 ]; then
```

```
    tar zxvf $1
    echo "done."
  fi
}
```

when you paste it into a JED buffer, you'll probably get:

```
untgz()  # Install a .tar.gz archive in current directory
{
  if [ $# != 0 ]; then
      tar zxvf $1
          echo "Done."
            fi
            }
```

This is caused by the `^M` key, normally bound to the `newline_and_indent` function. You may want to reserve a key to toggle between `newline_and_indent` and `newline`. Add this code to `.jedrc`:

```
static variable newline_indents = 0;
define toggle_newline_and_indent ()
{
  if (0 == newline_indents) {
    local_setkey ("newline_and_indent", "^M");
    newline_indents = 1;
    flush ("RET indents");
  }
  else {
    local_setkey ("newline", "^M");
    newline_indents = 0;
    flush ("RET does not indent");
  }
}

define global_mode_hook (hook_name)
{
  ...
  local_setkey ("toggle_newline_and_indent", "^Z^M");
}
```

Before pasting from other applications, you'll press `^Z RET` to disable indentation.

4.5 Reformatting Paragraphs

The command `M-X format_paragraph` reformats the current paragraph - that is, all the text until the following blank line. If you want the paragraph to be indented by, say, 5 positions, make the first line begin with 5 spaces, move the cursor to the first character, then issue the command.

If you'd like narrower paragraphs, set the variable `WRAP` in your `.jedrc`, or type `^X Esc` and enter `WRAP=`*nn*, where *nn* is the column where you want the text to wrap. To get unwrapped text, set `WRAP` to a very high value (say, 2000 or so).

To get the right margin aligned too, type `Esc 1 M-X format_paragraph`.

Modes may redefine paragraph delimiters, e.g. for avoiding reformatting comments of language constructs.

4.6 Keyboard Macros

Macros are used to define a set of commands that can be repeated at a later time. Use `M-X begin_macro`, `M-X end_macro`, and `M-X execute_macro` to define and run a macro.

If you need to enter some text while defining a macro, type `M-X macro_query`. The prompt `Enter String:` will appear in the mini-buffer. Any string that is entered will be inserted into the buffer, and the process of defining the macro continues. Every time the macro is executed, JED will prompt for a string to be inserted.

If you type `()=evalfile (■macro■)` at the S-Lang prompt, additional functions will become available. To assign a macro to a key, use `M-X macro_assign_macro_to_key`; to a function, `M-X macro_to_function`.

Saving and recalling macros is possible, if a bit tricky. Let's suppose that a macro was created that inserts `12345` and a newline in the current buffer. `M-X macrosavemacro` prompts you for a file name and a macro name. The file will look like:

```
%%%MACRO NAME: mymacro
@12345^M
```

To bind the macro to, say, `Ctrl-Z 1` and make it available in all modes, add this line to the `globalmodehook` function in your `.jedrc`:

```
define global_mode_hook (hook_name)
{
  ...
  local_setkey ("@12345^M", "^Z1");
  ...
}
```

4.7 Repeating Actions

The key sequence `Esc n`, where *n* is a digit, is used to repeat a command *n* times. For example, `Esc 25 x` inserts 25 'x' characters; `Esc 100 M-X execute_macro` runs the last macro 100 times; etc.

4.8 Rectangles

When you define a region, if the point and the mark are positioned on different columns *rectangle* is defined. Some function then let you perfomr actions on the rectangle.

You can use the functions `kill_rect` to cut the rectangle, `open_rect` to insert a blank rectangle, `copy_rect` to copy, `insert_rect` to paste, and `blank_rect` to blank-fill the rectangle.

4.9 Block Sorting

Let's suppose you want to sort a set of lines alphabetically or numerically. Begin by defining a rectangle on the data you want to sort, then type `M-X sort`. For example, let's consider this text:

```
Fruit:          Quantity:
lemons                 3
oranges               56
peaches              175
apples               200
pears                 37
```

To sort the data based upon the name: move the point on the `l` of `lemons`, set the mark, then move the cursor three positions after the `s` of `pears`. A rectangle is defined; type `M-X sort`. To sort the data by quantity, set the point two spaces before the `3` of the `lemons` line, then move the cursor after the `37` of the last line. This rectangle will sort the line upon the numbers.

Rectangles and block sorting are very useful when you work on multi-column data files.

S-Lang files: `sort.sl`

4.10 8-bit Characters and Mutekey

People whose language includes accented characters will benefit from `M-X digraph_cmd`, which lets the user compose such characters. To type special characters as you go, include a line like this in your `.jedrc`:

```
mute_set_mute_keys ("'`~^");
```

Pressing a quote, or a tilde, or a caret, followed by the character to accent, will insert an accented character in the buffer. Beware: this could make inserting quotes cumbersome in some situations, and won't work in all modes. If you get unexpected behaviour, type `M-X no_mode`, insert the problem characters, then revert to the mode you were using.

S-Lang files: `mutekeys.sl`

4.11 Abbreviations and Completion

Abbreviations are mode-dependent and work like in this example: you define that, say, "PS" stands for "PostScript" in all text-mode files; then toggle abbreviation mode; hence, each time you type "PS", it will be expanded to "PostScript". Abbreviations are saved in a file, usually `$HOME/.abbrevs.sl`.

To define your own abbreviations, you will edit that file by hand or type `M-X define_abbreviation`. You will be prompted for the word(s) you want to define an abbreviation for, and the abbreviation itself. Then type `M-X save_abbrevs` to save the abbreviations to the file, and `M-X abbrev_mode` to enable them.

To have abbreviations loaded at startup, see the example in Section Hooks (§ 8.5).

Completion is the automatic expansion of partially-typed words. If your text contains the word 'frobnication', typing 'fro' followed by `M-/` (`M-X dabbrev`) will either expand the whole word, or cycle amongst all possible completions.

S-Lang files: `abbrev.sl`, `dabbrev.sl`

4.12 Ispell

JED uses `ispell` for check spelling. If you're in doubt whether you misspelt a word, move the cursor over it and type `M-X ispell`. If the word is wrong, you will be prompted for alternatives.

Please note that only `ispell` works, `aspell` and other spell checkers do not!

S-Lang files: `ispell.sl`

4.13 Registers

Registers are 128 additional "clipboards". Select a region of text, then copy it to a register with `M-X reg_copy_to_register`. You will be prompted for a label to be associated with the register. To paste from a register, type `M-X reg_insert_register`; to view all active registers, `M-X register_mode`.

S-Lang files: `register.sl`

4.14 Tab Issues

By default, the `Tab` key is used in many modes to indent the text, but you can change the setkey definition in `.jedrc` to insert a real Tab character (ASCII 9). The `TAB_DEFAULT` variable specifies the tab size; for word processor-like tab editing, use `M-X edit_tab_stops`.

`M-X untab` operates on a region and converts tabs to spaces; `Esc 1 M-X untab` converts spaces to tabs.

S-Lang files: `tab.sl`, `untab.sl`

4.15 Regular Expressions

You may perform search and replace operations with regular expression (RE) support. The S-Lang library supports the following standard REs:

```
.                 match any character except newline
*                 matches zero or more occurences of previous RE
+                 matches one or more occurences of previous RE
?                 matches zero or one occurence of previous RE
^                 matches beginning of a line
$                 matches end of line
[ ... ]           matches any single character between brackets.
                  For example, [-02468] matches '-' or any even digit.
                  and [-0-9a-z] matches '-' and any digit between 0 and 9
                  as well as letters a through z.
\<                Match the beginning of a word.
\>                Match the end of a word.
\( ... \)
\1, \2, ..., \9   Matches the match specified by nth \( ... \)
                  expression.
```

In addition, the following extensions are also supported:

```
\c                turn on case-sensitivity (default)
\C                turn off case-sensitivity
\d                match any digit
\e                match ESC char
```

For example, to replace `"some text"` with `'some text'` (notice: from double quotes to simple quotes), you will search for `"\([a-zA-Z ]*\)"` and replace it with `'\1'`.

RE matching does not work across multiple lines. Moreover, JED's REs differ from `egrep`'s in the following aspects:

- the OR operator | is not supported;
- grouping operators `\(` and `\)` are not used to group REs to form a single RE. Thus, an expression such as `\(hello\)*` is not a pattern to match zero or more occurances of `hello` as it is in e.g., `egrep`.

S-Lang files: `regexp.sl`

4.16 Occur Search

`M-X occur` finds occurrences of regular expressions in the current buffer. A new buffer, called `*occur*`, is created; it contains the lines where the regexp was found. Switch to the `*occur*` buffer, then move the cursor to a line and press 'g'. In the other buffer, the cursor will be placed on the corresponding line.

S-Lang files: `occur.sl`

5 Buffer-related Issues

5.1 Window Splitting

The function `split_window` splits the current window in two; each window can display different buffers, or different parts of the same buffer. This is useful when you need to compare different parts of a file.

Use `M-X other_window` to move the cursor from one window to the other, and `M-X one_window` to get only one window on the screen.

5.2 Folding

Folding is a technique for hiding parts of a document, and is activated via the function `folding_mode`. The following functions/key bindings will be made available by this mode:

```
Function                Bound to key
-----------------------------------
fold_whole_buffer       ^C^W
fold_enter_fold         ^C>
fold_exit_fold          ^C<
fold_open_buffer        ^C^O
```

```
fold_fold_region        ^C^F
fold_open_fold          ^C^S
fold_close_fold         ^C^X
fold_search_forward     ^Cf
fold_search_backward    ^Cb
```

S-Lang files: `folding.sl`

5.3 Narrowing Regions

Once a region is defined, you can restrict editing on the region typing `M-X narrow`. Only the text in the region will be visible. To make the remaining text accessible again, type `M-X widen`.

5.4 Compiling

Let's suppose that you're writing C or Fortran code. Once the program is finished, you don't have to quit JED to compile it. Type `M-X compile` and provide a suitable command (e.g., `gcc -o foo foo.c`). If the source contains errors, they will be highlighted in the `*compile*` buffer; `M-X compile_parse_errors` and `M-X compile_previous_error` will move the cursor to the location of the next/previous error in the source buffer.

S-Lang files: `compile.sl`, `acompile.sl`

5.5 Tags

If you load a `tags` file made by the `ctags` program, you can visit the first occurrance of the tag under the cursor (or a tag you specify) typing `M-X find_tag`.

S-Lang file: `ctags.sl`

5.6 Commenting Code

When you write code, you can comment out regions of text with `M-X comment_region`. The function `uncomment_region` performs the opposite action.

S-Lang file: `comment.sl`

5.7 Buffer Editor

Mode designed for maintaining and editing open buffers; type `M-X bufed`. A new buffer called `*BufferList*` will be created. Type 'h' or '?' to get help.

S-Lang files: `bufed.sl`

5.8 Mail/Rmail (Unix Only)

Type `M-X mail` to write an email message; `M-X mail_send` to send it. In addition, if the executable `getmail` is included in the distribution, Jed acts as a MUA and gets mail from the standard mail spool. At this stage, it should be obvious how to use it. Main keys:

```
Key        Action
-------------------------
SPACE      scroll forward or select message
DELETE     scroll message backward
DOWN       move to next message (use space to select it)
UP         move to previous message
N          move to Next non-deleted message
P          move to Previously non-deleted message
D          Tag message for deletion
X          Really delete tagged messages and resequence folder
G          Get newmail
Esc 1 G    Prompt for a mail box and get new mail from that
Q          Quit this folder returning to top level (folder index)
T          Toggle headers. By default, jed will hide most of the
           headers. Use this key to unhide them.
O          Output message to a different folder. One will be
           created if necessary.
F          Forward message
R          Reply
M          mail
```

S-Lang files: `mail.sl`, `mailalias.sl`, `rmail.sl`, `sendmail.sl`

5.9 Man and Info Reader

UNIX users will like the built-in `man` and `info` readers; `M-X unix_man` and `M-X info_mode` to activate them. Man pages will be loaded in Most (§ **??**) mode.

S-Lang files: `man.sl`, `info.sl`

5.10 Quick Tips

- `M-X do_shell_cmd` executes a shell command and puts the resulting output in a buffer (`shell.sl`).
- `M-X toggle_readonly` is fairly obvious - it will make a buffer read-only or read/write.
- place the cursor on a parenthesis of any kind and type `M-X goto_match` to move the cursor to the matching parenthesis.
- `M-X cal` displays a calendar in a buffer (`cal.sl`).
- `M-X isearch_forward` and `M-X isearch_backward` perform incremental search operations (`isearch.sl`).
- `M-X replace_across_buffer_files` does a replace across all open buffers (`replace.sl`).
- `M-X trim_whitespace` removes all spaces after the last non-space character of a line.
- `M-X capitalize_word` changes a letter to upper case, while `M-X downcase_word` does the opposite.
- `M-X toggle_line_number_mode` toggles the visualisation of line numbers.

6 File-Related Issues

6.1 Recovering Files

JED can recover an interrupted session, like a system shutdown or a disconnected terminal. In such events, JED tries to auto-save its buffers. Files are saved with a leading and a trailing `#`, e.g. `#FileName.txt#`.

Whenever JED finds (loads) a file, it checks to see if an auto-save ile exists as well as the file's date. If the dates are such that the auto-save file is more recent, a message in the mini-buffer will alert the user. To recover your file, type `M-X recover_file`.

6.2 Editing Compressed Files

The `auto_compression_mode` function toggles this feature on or off. When it's on, files whose names end with `.gz`, `.Z`, or `.bz2` will be automatically uncompressed when read in, and compressed when written out. To enable this feature by default, include the following line in `.jedrc`:

```
auto_compression_mode ();
```

S-Lang file: `compress.sl`

6.3 File Locking (Unix Only)

Editing the same file in multiple JED sessions is potentially disruptive. File locking prevents a JED session from modifying a file that is already being edited in another session. Whenever a file gets modified, JED will attempt to lock the file. When the file is no longer modified (via getting saved to disk or changes discarded via undo), JED will unlock it.

6.4 End-Of-Line Issues

On DOS/Windows, text lines end with the two characters `^M^J` (ASCII 13 + ASCII 10), on UNIX with `^J`, on the Macintosh with `^M`. This difference makes it notoriously irksome editing UNIX text files under Windows, and vice versa. Jed solves this problem automatically when reading a file. Under UNIX, a 'C' in the status line indicates that DOS-style end-of-line is enabled; under DOS/Windows, an 'L' denotes UNIX-style end-of-line. You can change the end-of-line style with the command `M-X toggle_crmode`.

Macintosh text files will load as a long single line. To fix it, replace `^M` (enter it literally) with `^J`.

To have JED always save files with `^M^J`, add this line in `.jedrc`:

```
setbuf_info (getbuf_info () | 0x400);
```

6.5 Multiple Versions

In addition to the backup copy of the working file, you can get numbered copies that are created each time you save a buffer. These have a suffix like `.~3~` (third backup copy).

To enable multiple versions by default, add the entry `backups_on ();` in `.jedrc`. To disable this feature, type `M-X backups_off`.

S-Lang file: `backups.sl`

6.6 RCS and SCCS interface

JED provides access to the RCS versioning system. `M-X rcs_open_file` and `M-X rcs_read_log` prompt for the name of a file under RCS, such as `textfile.txt,v` or `RCS/textfile.txt,v`. To check a file in and out, use `M-X rcs_check_in_and_out`. When a file is checked out, it becomes write protected until you check it in again.

Similar commands are provided for SCCS: `M-X sccs_open_file` and `M-X sccs_check_in_and_out`.

S-Lang files: `rcs.sl`, `sccs.sl`

6.7 Directory Editor

This is a mode designed for maintaining and editing a directory and the files contained therein. Ypype `M-X dired`; you will be promted for a directory name, and a new buffer called `*dired*` will be created. Type 'h' to get help on the commands available.

S-Lang file: `dired.sl`

7 Modes

JED implements *modes* to facilitate the editing of program sources or markup languages like %LaTeX% or HTML. Typically, modes define a *syntax highlighting* scheme, provide *automatic indentation* and functions whose behaviour depends on *variables*. In some cases they provide a Mode menu entry, and often also provide a *hook* for customization purposes. You can change the default key binding and variables default values in your `.jedrc`.

For example, let's consider C mode. When you edit a file with extension `.c`, JED understands it's a C language file and turns C mode on. As you type your program, syntax elements are highlighted. Some mode-specific functions are now available from the Mode menu, along with new key bindings; you can change these by writing a function called `c_mode_hook` in your `.jedrc`.

Usually, JED recognises a file's associated mode by its extension. You can force JED to edit a file using a particular mode by putting this on its first line:

```
-*- your_mode -*-
```

For example, if you save your %LaTeX% files without the `.tex` extension, but want to edit them in `latex` mode, you will write `% -*- latex -*-` as the first line.

You may notice that modes are not very consistent with each other; that is, similar actions are often bound to very different key bindings. This will hopefully change in the future.

As of JED 0.99.18, supported modes and their features are outlined in the following table:

Mode	Syntax Highlighting	Indent	Font	Comment	Other
bibtex	y	y	y	n	n
c	y	y	n/a	y	y
c++	y	y	n/a	y	n
dcl	y	y	n/a	y	y
docbook	y	y	y	y	y
fortran 77	y	y	n/a	y	y

fortran 90	y	y	n/a	y	y
html	y	n	y	y	y
idl	y	y	n/a	n	y
java	y	y	n/a	y	y
latex	y	y	y	y	y
php	y	y	n/a	y	y
tex	y	n	n	n	y
lisp	y	y	n/a	n	n
lua	y	y	n/a	y	y
maple	?	?	n/a	?	y
matlab	y	y	n/a	y	y
nroff	y	n	n	n	n
perl	y	n	n/a	n	n
postscript	y	n	n/a	n	n
python	y	y	n/a	y	y
docbook	y	n	y	y	y
S-Lang	y	y	n/a	y	y
spice	y	n	n/a	n	n
tcl	y	y	n/a	n	n
tiasm	y	n	n/a	n	n
pascal	y	y	n/a	n	y
verilog	y	n	n/a	n	n
vhdl	y	n	n/a	n	n

Modes often provide new functions and new key bindings. In addition, virtually all modes provide hooks, which are explained in the next section.

7.1 Byte-Compiling Modes

Normal `.sl` files can be *byte-compiled*, i.e. turned to a more compact and faster format. The extension of byte-compiled S-Lang is `.slc`; many such files are created at install time.

If you want to byte-compile an S-Lang file in your `Jed_Home_Library`, move to that directory and start JED. Type this line in the `*scratch*` buffer:

```
byte_compile_file ("./your_file.sl", 0);
```

then type `M-X evalbuffer`. The file `your_file.slc` will be created.

7.2 Syntax Highlighting

JED provides two different kinds of syntax highlighting: normal and DFA. The latter is based on regular expressions, requires more memory, and makes JED a bit slower to start.

When a mode defines DFA highlighting, it may be convenient to create a *DFA cache*; this solves the problem of slow startup. The command

```
jed -batch -l preparse
```

creates DFA caches for all modes listed in `preparse.sl`, that is file with a `.dfa` extension.

Caveat: currently, DFA syntax highlighting breaks UTF8 Support (§ 8.1)!

7.3 C mode

This is a mode designed for editing C language files. After the mode is loaded, the hook `c_mode_hook` is called if it exists. Functions that affect this mode include:

```
Function              Default Binding
-------------------------------------
c_insert_bra          {
c_insert_ket          }
newline_and_indent    RETURN
indent_line           Tab
goto_match            ^-\
c_make_comment        Esc ;
c_format_Paragraph    Esc q
c_top_of_function     Esc ^A
c_end_of_function     Esc ^E
c_mark_function       Esc ^H
```

Variables affecting indentation include:

```
C_INDENT            % indentation of lines after '{'
C_BRACE             % how much to indent '{' if on a line by itself
C_BRA_NEWLINE       % '{' on a line by itself?
C_CONTINUED_OFFSET  % indentation of statements that continue on next line
C_Colon_Offset      % indentation of 'case' statements
C_Class_Offset      % indentation of members in a class declaration
```

You can set a predefined indentation style using using `c_set_style` and one of the following: `gnu linux jed bsd k&r`; e.g.

```
c_set_style ("gnu");
```

If you wish, you can make your own style using a statement like:

```
(C_INDENT, C_BRACE, C_BRA_NEWLINE, C_CONTINUED_OFFSET,
  C_Colon_Offset, C_Class_Offset) = (2,0,0,2,0,2);
```

S-Lang files: `cmisc.sl`

7.4 Fortran Mode

This is a mode designed for editing Fortran language files. After the mode is loaded, the hook `fortran_hook` is called if it exists. Useful functions include:

```
Function                       Default Binding
-----------------------------------------------
fortran_continue_newline        Esc RETURN
fortran_comment                 Esc ;
fortran_uncomment               Esc :
fortran_electric_label          0-9
fortran_next_statement          ^C^N
```

```
fortran_previous_statement      ^C^P
fortran_ruler                   ^C^R
fortran_beg_of_subprogram       Esc ^A
fortran_end_of_subprogram       Esc ^E
fortran_mark_subprogram         Esc ^H
```

Variables include:

```
Fortran_Continue_Char
Fortran_Comment_String
Fortran_Indent_Amount
```

S-Lang files: `fortran.sl`

7.5 LaTeX Mode

This is a mode designed for editing %LaTeX% files. After the mode is loaded, the hook `latex_mode_hook` is called if it exists. In addition, if the abbreviation table `latex` is defined, that table is used. Useful functions include:

```
Function                    Default Binding
-------------------------------------------
tex_insert_braces           ^C{
tex_font                    ^C^F
latex_environment           ^C^E
latex_section               ^C^S
latex_close_environment     ^C]
latex_insert_item           ^C^J
tex_comment_region          ^C;
tex_uncomment_region        ^C:
tex_comment_Paragraph       ^C%
tex_mark_environment        ^C.
tex_mark_section            ^C*
latex_toggle_math_mode      ^C~
tex_insert_macro            ^C^M
tex_complete_symbol         Esc Tab
tex_complete_symbol         Esc Tab
latex_help                  ^Ci
latex_indent_next_line      ^J
latex_indent_region         ^C^Q^R
latex_indent_section        ^C^Q^S
latex_indent_environment    ^C^Q^E
```

S-Lang files: `texcom.sl`, `latex.sl`, `latex209.sl`, `ltx-math.sl`.
Much better %LaTeX% modes are available on the net:

- http://www.ctan.org/tex-archive/support/jed/latex4jed/
- http://www.ctan.org/tex-archive/support/jed/jlm/

7.6 Tex Mode

This is a mode designed for editing TEX files. When `tex` mode is loaded, `tex_mode_hook` is called if it exists. Useful bindings:

```
Function             Default Binding
-----------------------------------
tex_insert_quote     "
tex_insert_quote     '
tex_blink_dollar     $
tex_ldots            .
```

S-Lang files: `texcom.sl`

7.7 HTML Mode

This is a mode designed for editing HTML files. The hook `html_mode_hook` is called if it exists.

If a region is defined (i.e., if a mark is set), many HTML tags will insert around the region, e.g. <B> and </B>.

If the variable `HTMLModeWraps` is set to 1, this mode will wrap (like `text_mode` does); otherwise, this mode won't wrap (like `no_mode`).

Keybindings are grouped according to function:

```
Key       Functions
------------------
^CA...    Anchors
^CD...    Definition lists
^CF...    Forms
^CH...    Headings, document type, etc.
^CI...    Images
^CL...    Lists
^CP...    Paragraphs styles, etc.
^CC...    Character styles
```

Additionally, some special movement commands and miscellaneous characters are defined:

```
Key     Action
--------------
^C^B    skip to beginning of prior HTML tag
^C^F    skip to end of next HTML tag
^C^N    mark next HTML tag from '<' to '>'
^C^P    mark prior HTML tag from '<' to '>'
^C&     insert HTML text for '&'
^C>     insert HTML text for '>'
^C<     insert HTML text for '<'
^CC     insert HTML comment (around region, if marked)
```

S-Lang files: `html.sl`

Again, another HTML mode is available:

- http://guido.gonzato.googlepages.com/html.sl

7.8 Docbook SGML Mode

This is a mode designed for editing Docbook SGML files. The hook `sgml_mode_hook` is called if it exists.

If a region is defined (i.e., if a mark is set), many SGML tags will insert around the region, e.g. <Emphasis> and </Emphasis>.

If the variable `WRAP_INDENTS` is set to 1, this mode will indent the text. Also, `SGML_INDENT` contains the number of characters of indentation (default 2).

All tags are inserted via the Mode menu. Additionally, some special movement commands and miscellaneous characters are defined:

```
Key     Action
--------------
^C^B    skip to beginning of prior SGML tag
^C^F    skip to end of next SGML tag
^CP     insert <Para>
^C&     insert SGML text for '&'
^C$     insert SGML text for '$'
^C>     insert SGML text for '>'
^C<     insert SGML text for '<'
^C.     insert SGML text for '...'
^C;     insert SGML comment (around region, if marked)
```

S-Lang files: `docbook.sl`

Similar to Docbook SGML mode is Linuxdoc mode, available at

- http://guido.gonzato.googlepages.com/linuxdoc.sl

7.9 Most Mode

This is a simple mode used for browsing text files, which will be opened read-only. Press `Space` to advance a page, `Bs` to go back, `/` to search forward, `?` to search backwards. Press `h` to get a short help message.

S-Lang files: `most.sl`

8 Advanced Customisation

JED can be fully customised writing S-Lang code in your `.jedrc` and/or in external files. You can get started with S-Lang simply studying the `.sl` files in the JED distribution. Full documentation is available in the S-Lang source package.

8.1 UTF8 Support

If the environment variable `JED_UTF8` exists and is 1, JED runs with UTF8 encoding. Beware though: currently, DFA syntax highlighting is not compatible with it. For instance, `à ç ì` are displayed as <E0> <E7> <EC>: that is, the hex character codes.

8.2 Setting Keybindings

Consider this S-Lang code:

```
unsetkey (^H);
unsetkey (^N);
setkey ("insert (string(what_line))", "^H");
setkey (" Guido Gonzato, Ph.D.", "^N");
```

The first two lines unset previous definitions of `^H` and `^N`. The `^H` key is bound to the function `insert (string(what_line))`. The effect of `^N` is different, because the text given as first argument of `setkey` starts with a space. In this case, the text itself is inserted, not interpreted as S-Lang code.

`setkey` applies to the global keymap (i.e., in all modes). To restrict a key binding to a specific keymap, use `definekey`:

```
unsetkey ("^N", "LaTeX-Mode");
definekey (" \LaTeX{}", "^N", "LaTeX-Mode");
```

Finally, `local_unsetkey` and `local_setkey` behave like `(un)setkey`, but apply to the current keymap only.

8.3 Loading Additional `.sl` Files

Writing lots of code in `.jedrc` is not practical; a better method is to write it in an external `.sl` file. Continuing the example above, let's suppose you write key definitions to `mykeys.sl`. Add this file to the JED library and add this line in `.jedrc`:

```
() = evalfile ("mykeys");
```

8.4 Autoloading Functions

Sometimes, using `evalfile` is not the best method of loading functions. Let's suppose you *occasionally* want to use the functions `fun1`, `fun2`, `fun3` defined in the file `functions.sl`. `evalfile` will load a given file even if its functions are not actually used, which is a waste of computer resources.

Another method of loading functions is via the `autoload` function. `autoload` is used to declare a function to the interpreter, and indicate that it should be loaded from a file when it is actually used.

For example, defining these lines in `.jedrc`:

```
autoload ("fun1", "functions");
autoload ("fun2", "functions");
autoload ("fun3", "functions");
```

will cause JED to load `functions.sl` when one of its functions is required.

8.5 Hooks

A *hook* is a user defined S-Lang function that may be used to extend the editor or to modify how it behaves under certain circumstances. There are two kinds of hooks that vary according to whether the hook is called "internally" from the editor by the underlying C code, or whether the hook is called from another S-Lang function.

The hooks may be subdivided further according to their scope. They may apply to all buffers (globally), only buffers sharing the same mode (modal), or to a single buffer (local). Please read `hooks.txt`.

An example of hook is the following function:

```
define latex_mode_hook ()
{
  set_abbrev_mode (1); % use LaTeX-specific abbreviations
  if ( () = abbrev_table_p ("LaTeX") )
    use_abbrev_table ("LaTeX");
  local_setkey (" \\‘a", "à"); % pressing ’à’ inserts ’\‘a’
  local_setkey (" \\‘e", "è");
  local_setkey (" \\’e", "é");
  local_setkey (" \\‘\\i{}", "ì");
  local_setkey (" \\‘o", "ò");
  local_setkey (" \\‘u", "ù");
}
```

This hook is called whenever `latex` mode is entered. It loads %LaTeX%-specific abbreviations and lets the user type accented characters, expanding them to the right key sequences.

---

9 The End

This quick reference was written by Guido Gonzato, Ph.D. `guido dot gonzato at poste dot it`. Contributions by John E. Davis.

Many thanks to John for writing JED!

Please drop me a line if you think the information contained in this document is inaccurate or unclear.