GNU Emacs REDUCE IDE

An Integrated Development Environment for REDUCE:
Major modes for editing and running REDUCE source code
Software version 1.5

Francis J. Wright (`https://sourceforge.net/u/fjwright`)

Manual last updated Time-stamp: <2017-11-29 18:32:35 fjw>

This manual is for REDUCE IDE (version 1.5, updated Time-stamp: <2017-11-29 18:32:35 fjw>), which provides GNU Emacs major modes for editing and running REDUCE source code.

Table of Contents

1 Introduction to REDUCE IDE 1

2 Installation of REDUCE IDE 2

3 General features of REDUCE edit mode 2

4 Statement-oriented commands 3

5 Procedure-oriented commands 5

6 Support for REDUCE comments 6

7 Indenting REDUCE code automatically 8

8 Templates for REDUCE structures 10

9 Keyword completion and abbreviation expansion 11

10 Font-lock support for automatic font selection 11

11 Access to procedures and operators 13

11.1 Show procedure mode 13
11.2 Imenu support 13
11.3 Support for tag files 14

12 Miscellaneous minor features and bugs 14

12.1 Groups and blocks: delimiter highlighting and moving over 14
12.2 Major mode menu 15
12.3 REDUCE mode version information 15
12.4 Known and potential problems 15
12.5 Special function keys 15

13 Customization of REDUCE IDE 16

13.1 REDUCE mode interface customization 16
13.2 REDUCE mode format & display customization 16
13.3 REDUCE mode hooks 17

14 Running REDUCE in a buffer 18
14.1 Introduction to REDUCE run mode 18
14.2 Installation of REDUCE run mode 19
14.3 Running PSL REDUCE on Windows 19
14.4 Customization of REDUCE run mode 19
14.5 Running, re-running and switching to REDUCE 20
14.6 Running complete REDUCE programs 21
14.7 Inputting code fragments to REDUCE 22
14.8 Inputting, compiling and loading REDUCE files 22
14.9 Run mode key bindings and menu 22

15 Feedback: bug reports, suggestions, comments, ... 23

Command Index 25

Variable Index 25

Keystroke Index 26

Concept Index 27

1 Introduction to REDUCE IDE

This manual documents the GNU Emacs Integrated Development Environment (IDE) for REDUCE, which comprises a primary major mode for syntax-directed editing of REDUCE source code (edit mode) and a subsidiary major mode for running REDUCE as an inferior process with input and output via a buffer (run mode). REDUCE is a system and language for algebraic computing developed originally by Anthony C. Hearn, which is now Open Source and available from SourceForge (`https://sourceforge.net/projects/reduce-algebra/`). It therefore shares the GNU spirit of collaborative software development, which provided part of my motivation to begin this project. REDUCE is also written in Lisp, as is (most of) Emacs. However, the REDUCE user language is similar to Algol-60 (which looks a bit like a cross between Pascal and FORTRAN-77).

Development of REDUCE edit mode began tentatively in late 1992, and serious development of the current version began in early 1994, and continued sporadically until 2001. Development of REDUCE run mode began in late 1998. The latest released versions of the software and documentation are available from SourceForge (`https://sourceforge.net/projects/reduce-algebra/`). Comments, suggestions, bug reports, etc. are welcome.

The current version of the REDUCE IDE is intended for use with GNU Emacs version 25 and later, which I will endeavour to support under recent versions of Microsoft Windows and Ubuntu Linux; it might run under closely related versions of GNU Emacs but I cannot explicitly support either XEmacs or other platforms.

REDUCE IDE version 1.5 provides two major updates. One is a substantial rewrite of REDUCE run mode to provide explicit support for both CSL and PSL REDUCE, and for easily running multiple copies of REDUCE simultaneously. The other is distribution as a multi-file package via the REDUCE IDE web page (`http://reduce-algebra.sourceforge.net/reduce-ide/`), which makes installation and updating very easy via the GNU Emacs package manager (see Section "Emacs Lisp Packages" in *GNU Emacs Manual*).

However, beware that in native Microsoft Windows GNU Emacs (only) PSL REDUCE currently does not run correctly without some additional support, which is explained below. The problem appears to be that it needs to be explicitly run from a shell. I hope to improve this situation in a later version of REDUCE IDE.

This manual assumes that you are familiar in general with both Emacs and REDUCE.

The purpose of REDUCE edit mode is to provide editing commands that are aware of the syntax of the REDUCE language, and therefore allow operations to be performed on the major syntactic elements, namely statements, procedures and comments. To the reader who has never used a syntax-directed editor, I can only say that it is surprisingly useful! In particular, the automatic indentation code provides valuable clues to potential REDUCE programming errors by showing how the REDUCE parser is likely to interpret the code (see Chapter 7 [Indenting REDUCE code automatically], page 8).

The purpose of REDUCE run mode is to provide a friendly interface to a **command-line version** of REDUCE running as an inferior process in an Emacs buffer. REDUCE run mode inherits much of its functionality from REDUCE edit mode and cannot be run alone. The assumption is that normal use will involve editing one or more REDUCE source files and running REDUCE simultaneously, and this is what REDUCE run mode aims to support.

All REDUCE IDE commands are self-documenting as usual in Emacs, including in particular the modes themselves. Hence, for an overview of the version of a REDUCE mode that is actually installed in your Emacs, select it in some buffer and then give the command `C-h m` (`describe-mode`) or use the *Help* menu option *Describe*.

2 Installation of REDUCE IDE

REDUCE edit mode is provided by a file called `reduce-mode.el`, which is a file of ELisp source code. This file should be byte-compiled, and the compiled file `reduce-mode.elc` should be installed in the normal directory from which Emacs loads its Lisp code (if you have suitable permission). Otherwise, customize your Emacs `load-path` so that Emacs can find `reduce-mode.elc`. See Section "Customization" in *The Emacs Editor*.

Emacs initialization and customization is stored in a file that is normally called `.emacs` and lives in your home directory. The precise meaning of "home directory" depends on both your OS and Emacs version; the easiest way to find it in Emacs is to visit the directory `~`, or just visit the file `~/.emacs` directly. Your `.emacs` file is updated automatically by the Emacs customization facility and can also be edited by hand to add other configuration. See Section "Init File" in *The Emacs Editor*.

Before REDUCE mode can be used, the file `reduce-mode.elc` must be loaded. This is necessary only once per Emacs session. It can be loaded explicitly, most easily by giving the command *M-x load-library reduce-mode*. However, you will probably want `reduce-mode.elc` to be loaded automatically the first time you (explicitly or implicitly) turn on REDUCE mode. The way to do this, on a per-user basis, is to put the following statement into your `.emacs` file:

```
(autoload 'reduce-mode "reduce-mode"
          "Major mode for REDUCE code editing" t)
```

This statement is completely innocuous and will have no effect unless the user selects REDUCE mode. It could therefore quite safely be put in a system-wide configuration file (e.g. `default.el` or `site-start.el`). See Section "Init File" in *The Emacs Editor*.

It is also very convenient to have REDUCE mode turned on automatically when editing a REDUCE source code file. This can be done based on the "extension" of the filename. Provided you end all REDUCE source code file names with the standard extension `.red`, the following statement in your `.emacs` file will have the desired effect:

```
(add-to-list 'auto-mode-alist '("\\.red\\'" . reduce-mode))
```

Other extensions can be used as well or instead; if you use a different file naming convention then make the appropriate change(s) to the above statement. Emacs also provides other facilities that can be used for controlling major modes.

Installation of REDUCE run mode is documented separately. See Chapter 14 [Running REDUCE in a buffer], page 18.

3 General features of REDUCE edit mode

REDUCE edit mode can be selected by giving the command *M-x reduce-mode*, although normally it will be selected automatically, probably via the filename extension (see Chapter 2 [Installation of the REDUCE IDE], page 2).

The commands provided by REDUCE mode are aware of REDUCE syntax and ignore the contents of strings and the case of characters. Except for the special comment commands (see Chapter 6 [Support for REDUCE comments], page 6), they also ignore comments. The standard GNU Emacs indentation (see Chapter 7 [Indenting REDUCE code automatically], page 8) and comment commands are supported, either via the general Emacs mechanisms or by re-binding the standard keys to REDUCE-mode versions of standard commands. The design of this mode is modelled primarily on Lisp mode and at present the comment conventions basically follow those of Lisp mode, except that comments are started by percent (`%`) signs or the keyword `comment`. It has also taken some ideas from FORTRAN mode.

The standard Emacs syntax tables have been modified to reflect REDUCE syntax, so that for example Emacs knows that the REDUCE escape character is `!`. However, there remain some unresolved problems concerning the REDUCE escape character. See Chapter 10 [Font-lock support for automatic font selection], page 11. See Chapter 12 [Miscellaneous minor features and bugs], page 14.

Blank lines separate "paragraphs".

Loading the REDUCE mode library runs any functions on `reduce-mode-load-hook`, which can be used to customize global features of REDUCE mode such as its key map. Entry to REDUCE mode runs any functions on `reduce-mode-hook`, which can be used to customize buffer-local features of REDUCE mode, e.g. to turn on font-lock mode. See Chapter 2 [Installation of the REDUCE IDE], page 2. See Chapter 13 [Customization of the REDUCE IDE], page 16.

REDUCE mode is intended to support both the algebraic and symbolic modes of REDUCE. It provides very limited support for Lisp syntax to the extent that it is likely to be used in symbolic-mode code, and hence it understands the significance of the quote symbol (`'`) to some extent. Syntax-directed editing naturally works correctly only if the syntax of the source code being edited is correct. If it is not then strange things can happen, and the services of the Emacs undo facilities may be required!

DEL
M-x backward-delete-char-untabify
: Delete one character backwards, converting tabs to spaces if necessary.

A major mode menu provides convenient access to most of the major facilities of REDUCE mode.

4 Statement-oriented commands

The most basic facility provided by REDUCE mode is the ability to move forward and backward by statement through a file of REDUCE source code. Moving by one statement means moving to the beginning or end of the *logical* statement currently containing (or respectively preceding or following) point, which may involve skipping many actual statements that are contained within the current statement. In particular, as Emacs looks for the beginning or end of a statement it will skip complete compound statements (`begin ... end`), group statements (`<< ... >>`), and bracketed expressions (`(...)`, `{...}` and `[...]`, even though square brackets are not normally used in REDUCE). Bracket skipping is controlled entirely by the Emacs syntax table.

Hence, "statement" in this manual will normally mean a complete *logical statement*. A syntax-directed editor clearly must perform a limited amount of parsing, but it must be remembered that a syntax-directed editor has the following important differences from a normal parser, because their basic purposes are different:

- A syntax-directed editor must be able to parse both forwards *and backwards*.
- It will typically parse only locally for speed and must therefore parse based on incomplete information.
- It is provided for the convenience of the user and therefore need not obey precisely the full syntax of the language, provided it is consistent and reliable.

Emacs considers REDUCE statements to be terminated by the statement separator characters `;` and `$`. It also considers statements contained within any kind of brackets to be terminated by those brackets, statements within compound statements (`begin ... end`) to be terminated by the `begin` and `end` keywords, and statements within group statements (`<< ... >>`) to be terminated by the `<<` and `>>` tokens. Commas are not considered to separate statements.

More precisely, a statement is considered to begin at the first non-white-space character following the previous statement separator, opening bracket, `begin` or `<<`. It is considered to end immediately after the first statement separator or immediately after the last non-white-space character preceding a closing bracket, `end` or `>>`.

The current philosophy of REDUCE mode is that the statements within compound or group statements form essentially isolated systems, and that the basic statement-oriented commands should not move point either into or out of this system. Separate commands are provided to move into and out of compound and group statements. However, if you try hard enough, REDUCE mode will let a simple statement-oriented command move out of (but never into) a compound or group statement. Trying hard enough means repeating the same command enough times, which is determined by the value of the user option `reduce-max-up-tries` (see Chapter 13 [Customization of the REDUCE IDE], page 16), which currently has the default value 2. The overall effect of this is to enforce a brief pause (one ineffective command execution) that serves to prevent you from skipping out of a compound or group statement accidentally, but without causing any serious inconvenience.

There is special code to handle the keyword `end` when it is used as the end-of-file marker. The following commands all accept a numerical argument, which defaults to 1. The commands to move forward or backward by statements do not move in the opposite direction if given a negative argument, in which case they do not move at all.

C-c C-n
M-x reduce-forward-statement
: Move forward to the end of the statement. With an argument, do it that many times. If looking at the end of a block or group, or the end-of-file marker, move over it after `reduce-max-up-tries` consecutive interactive tries.

C-c C-p
M-x reduce-backward-statement
: Move backward to the start of the statement. With an argument, do it that many times. If looking at the beginning of a block or group move over it after `reduce-max-up-tries` consecutive interactive tries. The end-of-file marker is treated as a statement.

C-c C-k
M-x reduce-kill-statement
: Kill the rest of the current statement; if there are no non-blank characters kill through the newline. With an argument, kill that many statements from point. Negative arguments kill complete statements backwards, cf. `kill-line`.

C-c C-u
M-x reduce-up-block-or-group
: Move backward up one level of block or group, i.e. to the beginning of the nearest unpaired `begin` or `<<`. A universal argument means move forward, i.e. to the end of the nearest unpaired `end` or `>>`. With a numeric argument, do it that many times, where a negative argument means move forward instead of backward.

C-c C-d
M-x reduce-down-block-or-group
: Move forward down one level of block or group, i.e. to the end of the nearest unpaired `begin` or `<<`. A universal argument means move backward to the beginning of the nearest unpaired `end` or `>>`. With a numeric argument, do it that many times, where a negative argument means move backward instead of forward.

5 Procedure-oriented commands

Files of REDUCE source code frequently consist mainly of procedure definitions. This is certainly true of symbolic-mode code, and hence it is true of most of the source code of the REDUCE system itself. REDUCE mode provides the following operations on procedures. They work on all kinds of REDUCE procedures provided they contain the keyword `procedure` somewhere within the first statement of their definition.

A procedure is considered to begin at the first non-white-space character of the definition, and to end after the statement defining the procedure body. White space and the first newline after the procedure body are always considered to be part of the procedure. The commands to mark, kill and reformat a procedure also include *all* blank lines after the procedure definition, because this seems most convenient in practice. Some procedure-oriented commands support a prefix argument.

The two commands for moving over procedures accept a positive integer argument that indicates by how many procedures to move – the default is 1. These commands do not move in the opposite direction if given a negative argument, in which case they do not move at all.

C-M-e
M-x reduce-forward-procedure
: Move forward to the next end of a procedure. With a numeric argument, do it that many times.

C-M-a
M-x reduce-backward-procedure
: Move backward to the next start of a procedure. With a numeric argument, do it that many times.

Regardless of whether point is within a procedure or not, these two commands move respectively to the first following end of a procedure, or the first preceding start of a procedure. To move to the start of the next procedure, move forward to its end and then move backward to its start.

The remaining commands do not accept an argument because (even without an argument) they can change large portions of text. Marking a procedure is the basis of the other operations on procedures.

C-M-h
M-x reduce-mark-procedure
: Put the mark after the next end of a procedure and point at the start of that procedure. A procedure ends *after* any trailing white space. With a numeric argument, mark that many following procedures including this one.

C-c k
M-x reduce-kill-procedure
: Kill the procedure (and trailing white space) ending after point.

C-M-q
M-x reduce-indent-procedure
: Indent the procedure (and trailing white space) ending after point. See Chapter 7 [Indenting REDUCE code automatically], page 8.

It is often desirable to be able to see as much as possible of a procedure definition within the current window. The standard Emacs command `reposition-window` (see Section "Scrolling" in *The Emacs Editor*) attempts to do this for Lisp functions, and the command

`reduce-reposition-window` provides a harness to apply this function to REDUCE procedures, to which the standard key *C-M-l* is rebound.

C-M-l
M-x reduce-reposition-window
: Reposition the procedure containing point to maximize its visibility within the window. See Section "Scrolling" in *The Emacs Editor*, and see the documentation for the function `reposition-window` for details.

To restrict all editing to a single REDUCE procedure, the standard Emacs key *C-x n d* that runs the command `narrow-to-defun` is rebound to a function to narrow to the current procedure.

C-x n d
M-x reduce-narrow-to-procedure
: Make text outside the current procedure invisible. The procedure visible is the one that contains point or follows point. With a prefix argument, narrow to the following arg procedures including this one. See Section "Narrowing" in *The Emacs Editor*.

6 Support for REDUCE comments

There are two comment conventions used in REDUCE. One is the comment statement, which is a statement that begins with the keyword `comment` and ends with a statement separator. This is not used much in modern REDUCE code. The most commonly used form of comment begins with a `%` character and ends at the end of the line. Hence, it can appear either on its own on a line or at the end of a line after other code.

Comments are ignored (skipped) by all syntax-directed commands. (This is not trivial to achieve, since comments can contain essentially arbitrary text including keywords, and %-comments can contain statement separators, which do not have any syntactic significance.) There is currently no way to use any of the REDUCE syntax-directed commands on comment statements.

There is no special support for comment statements other than that they are currently *completely* ignored. There is considerably more support for %-comments, much of which is already built into Emacs because %-comments are very similar to the comments used in Emacs Lisp. Indeed, the comment conventions supported by REDUCE mode are modelled primarily on those used in Emacs Lisp mode.

The comment commands are intimately related to the automatic comment indentation conventions. (These are the indentation conventions enforced by the Emacs comment and indentation commands, although the user is not otherwise forced to follow them.) See Chapter 7 [Indenting REDUCE code automatically], page 8.

The indentation of a %-comment that begins with no more than 2 `%` characters together and appears alone on a line is determined by the previous non-blank line. If this is a procedure (header) statement then the comment line is indented relative to it, otherwise it has no indent relative to the previous line, and at the beginning of a file it is not indented at all. A %-comment at the end of a line of code is indented to the column specified by the value of the standard Emacs buffer-local variable `comment-column`, which by default is 40 (half way across a "standard" 80 column page), unless the code extends beyond this column. In that case, the comment begins one space later.

This convention can be over-ridden as follows. If the comment begins with 3 or more `%` characters then the comment indentation is not changed. This allows a comment to be placed anywhere on an empty line without any risk of it being automatically re-indented.

A new single-%-comment can be introduced and/or automatically indented by the standard Emacs command `indent-for-comment`, normally bound to the key *Meta-;*. An existing %-comment can be automatically continued on the next line by the standard Emacs command `indent-new-comment-line`, normally bound to the key *Meta-LFD*. This will copy the structure of the %-comment to be continued, including the number of `%` characters and the indentation. All other indentation commands will also indent %-comments, in particular those bound to the `TAB` and `BACKTAB` (i.e. *Shift-TAB*) keys. See Chapter 7 [Indenting REDUCE code automatically], page 8.

M-;
M-x indent-for-comment
: Indent this line's comment appropriately, or insert an empty comment.

M-LFD
M-x indent-new-comment-line
: Break the line at point and indent, continuing a comment if presently within one. The body of the continued comment is indented under the previous comment line.

The only program text that it normally makes sense to fill or justify is comment text. Hence, REDUCE mode rebinds the key *M-q* that normally fills or justifies a paragraph to the command `reduce-fill-comment`. This should be completely safe to use in REDUCE code (unlike `fill-paragraph` etc., which would be a potential disaster were there no undo facility!), and makes it easy to keep comments formatted tidily. Currently this command fills or justifies only %-comments and not comment statements.

M-q
M-x reduce-fill-comment
: Fill successive %-comment lines around or immediately following point. A prefix argument means justify as well.

REDUCE mode also provides commands for turning sections of text into %-comments by adding `%` characters at the start of each line, which will be referred to as "start-comments". These commands are intended primarily for temporarily preventing REDUCE from executing sections of code without actually removing them. Such a section can be either the current region or the procedure ending after point. By default, these commands automatically toggle the comment status. When given an interactive argument, they remove any start-commenting of the specified section of text if the argument is negative (or null) and insert start-commenting if the argument is positive. The precise text that is added to or removed from each line is the value of the variable `reduce-comment-region-string`, which defaults to '`%% `'.

C-c ;
M-x reduce-comment-region
: Comment/uncomment every line in the region. By default, it toggles the commenting, i.e. it comments the region if it is uncommented and uncomments if it is commented. With an interactive argument, comment if non-negative, uncomment if null or negative (cf. minor modes). When commenting, it puts the value of the variable `reduce-comment-region-string` at the beginning of every line in the region.

C-c :
M-x reduce-comment-procedure
: As for `reduce-comment-region`, but applies to the procedure ending after point.

7 Indenting REDUCE code automatically

Indentation refers to the white space at the left of a line, which therefore determines the column in which the actual text of the line begins. Indentation is used in normal English text to indicate the beginning of paragraphs, quotations, lists, etc. and hence to indicate the logical structure of a document.

It is very important to use systematic indentation to indicate the logical structure of the source code of a computer program. Whilst the general principles of indentation are largely agreed, precise indentation conventions vary from author to author. The automatic indentation currently provided by REDUCE mode is very inflexible and reflects very much my own style of indentation. Future versions may provide more flexible and customizable indentation.

Currently all indentation is done in steps consisting of a fixed number of columns determined by the value of the variable `reduce-indentation` (see Chapter 13 [Customization of the REDUCE IDE], page 16), the default value of which is 3. This is the indentation recommended by A. C. Hearn (the principal author of REDUCE) for the indentation of the first line after a procedure (header) statement.

REDUCE mode provides fairly intelligent automatic indentation. The style used is as follows, where the indentation of a child statement is expressed relative to the parent statement. Each top-level statement is indented to the left margin. Procedure bodies are indented by one step. Bodies of multi-line compound and group statements are indented by one step and labels are exdented to match the beginning of the enclosing block. Lines that begin with `end` or `>>` are exdented to match the line containing the matching `begin` or `<<`. Bodies of control structures and lines that continue a previous statement are indented by one step. As parts of larger statements, compound and group statements themselves are generally not indented if they occupy multiple lines (because their bodies are indented) but they are indented if they occupy only a single line.

When a new line that is empty is being indented, the indentation can be based only on the preceding code, and not on the code that will appear in the line. Therefore, it is often necessary to re-indent a line in order to get consistent indentation. This seems a little strange, but it is unavoidable (given the syntax of REDUCE and the indentation style that I have chosen). It is for this reason that the key `LFD` runs the command `reindent-then-newline-and-indent` rather than just `newline-and-indent`. But see also below.

The command to indent, or re-indent, a line of text is `reduce-indent-line`, normally bound to the key `TAB`. If re-run immediately after itself (or run immediately after `reindent-then-newline-and-indent` (`LFD`) or `newline-and-indent` (*`M-x newline-and-indent`*)) then it indents by one further step. This is non-standard additional flexibility provided by REDUCE mode. To force a line back to its standard indentation after multiple use of the `TAB` key, simply execute any other command(s) and then press `TAB` *once*. The execution of `reduce-indent-line` is independent of the position of point within the line. It does not move point relative to the text around it unless point was within the indentation, in which case it is left before the first non-blank character (i.e. at the end of the indentation), or at the end of the line if it is empty. Normally, however, the most convenient way to use automatic indentation is to terminate each line of code with `LFD` rather than `RET`. See Chapter 12 [Miscellaneous minor features and bugs], page 14.

When called with any argument, `reduce-indent-line` will indent the current line correctly and then re-indent the rest of the logical statement containing point by the same amount that the current line was re-indented. This is *not* the same as correctly re-indenting the subsequent lines – it re-indents them rigidly, without changing their relative indentations at all, and is much faster.

TAB
M-x reduce-indent-line
: Indent or re-indent the current line as REDUCE code. Indents to a fixed style determined by the current and previous non-blank lines. Subsequent consecutive calls indent additionally by `reduce-indentation`. With an interactive argument, indent any additional lines of the same statement rigidly together with this one.

LFD
M-x reindent-then-newline-and-indent
: Re-indent the current line, insert a newline, then indent the new line. Indentation of both lines is done using `reduce-indent-line`, which is bound by default to *TAB*.

With the current indentation style, it is not possible in all cases to determine the correct indentation until after some text has been entered on a line. This applies to the terminal delimiter of a block `end` or group `>>` when it appears alone on a line and to an `else` clause. Therefore, REDUCE mode can automatically re-indent the current line once there is enough text to recognise that this is necessary. It does this only when it is otherwise idle and only when the relevant text has just been typed. It is not done if the cursor is later moved onto such a line since it is assumed that the desired indentation has been set by then. (The indentation of any text can, of course, be changed at any time, but it will never be automatically changed retrospectively!) This facility is turned on and off by the command `reduce-auto-indent-mode`, the length of idle time required before the facility will operate is controlled by the user option `reduce-auto-indent-delay`, and whether the current line is auto-indented by this facility is controlled by the regular expression that is the value of the user option `reduce-auto-indent-regexp`. Auto-indentation is on by default. See Chapter 13 [Customization of the REDUCE IDE], page 16.

M-x reduce-auto-indent-mode
: Toggle REDUCE Auto Indent mode. With a prefix argument, turn the mode on if and only if the argument is positive. When REDUCE Auto Indent mode is enabled, after `reduce-auto-indent-delay` seconds of Emacs idle time re-indent the current line if the text just typed matches `reduce-auto-indent-regexp`.

A section of code can be re-indented using one command if it is first marked as the current region, or the whole buffer or a complete procedure definition can be re-indented by a single command. The latter command works by marking the procedure and then re-indenting the region; it currently leaves the procedure marked. Region (and hence procedure) indenting is currently implemented inefficiently by applying the single-line indentation algorithm line-by-line, and hence is very slow for a large region or procedure. In some future version it may be re-implemented more efficiently.

C-M-
M-x reduce-indent-region
: Indent or re-indent the region as REDUCE source code by applying `reduce-indent-line` to each line. With a prefix argument it indents the whole buffer.

C-M-q
M-x reduce-indent-procedure
: Indent or re-indent the procedure (and trailing white space) ending after point by applying `reduce-indent-line` to each line.

An inverse of the extra-indentation facility is provided to decrease the indentation by one step. This command is bound to *Shift-TAB* (i.e. `BACKTAB`) if possible, but not all platforms support this (because it has no ASCII representation). It may be no different from `TAB` alone, or it may generate an obscure ASCII sequence, so just try *Shift-TAB*, or ask Emacs what function is bound to *Shift-TAB* (by using *C-h k*). A default key binding, which should work on all

platforms, is provided as *C-c TAB*. (However, currently this is rebound by REDUCE run mode. See Section 14.9 [Run mode key bindings and menu], page 22.)

With an argument, `reduce-unindent-line` rigidly unindents by one step the current line and the rest of the logical statement as an inverse of extra applications of `reduce-indent-line` with an argument.

BACKTAB
Shift-TAB
C-c TAB
M-x reduce-unindent-line
: Unindent the current line as REDUCE code by deleting `reduce-indentation` spaces from the beginning of the line. With an interactive argument, unindent any additional lines of the same statement rigidly along with this one.

8 Templates for REDUCE structures

Commands are provided to insert and format the major REDUCE language structures; currently block or compound (`begin ... end`), group (`<< ... >>`) and conditional (`if ... then ... else ...`) statements are supported. By default they are formatted to be multi-line. If given a prefix argument, the commands to insert block and group statements (composites) format them on a single line (appropriate in some very simple cases).

If there is text on the line after where a composite is inserted then it is moved into the body of the composite; if transient mark mode is on and the mark is active then the whole region is moved into the composite; the composite is then re-indented.

The cursor is left in place to enter the body statements of a group, whereas a block is inserted complete with an empty `scalar` declaration and the cursor is left in place to enter the names of the scalar variables.

C-c b
M-x reduce-insert-block
: Insert and indent a `begin scalar ; ... end` block and position point inside. With an argument put `begin` and `end` on the same line.

C-c <
M-x reduce-insert-group
: Insert and indent a `<< ... >>` group and position point inside. With an argument put `<<` and `>>` on the same line.

C-c i
M-x reduce-insert-if-then
: Insert `if ... then` and position point inside. With argument include a correctly indented `else` on a second line.

Probably the easiest way to access these templates from the keyboard is not directly as described above but via the generalized completion facilities described in the next chapter. See Chapter 9 [Keyword completion and abbreviation expansion], page 11.

9 Keyword completion and abbreviation expansion

Emacs provides various standard facilities for semi-automatic completion of key words and phrases (see Section "Completion for Symbol Names" in *The Emacs Editor*, see Section "Dynamic Abbrev Expansion" in *The Emacs Editor*). REDUCE mode provides completion of common REDUCE key words and phrases, such as '`procedure`', by typing the first few letters of a key word or phrase and then pressing *Meta-TAB*. (For use under Microsoft Window see Chapter 12 [Miscellaneous minor features and bugs], page 14) This works in a similar way to completion in other major modes (see Section "Completion for Symbol Names" in *The Emacs Editor*).

REDUCE mode also provides *abbreviations* that are expanded like completions, except that they are *replaced* by their expansions rather than completed. Examples of the abbreviations currently defined are:

```
("ap" . "algebraic procedure ")
("st" . "such that ")
("sop" . "symbolic operator ")
("sp" . "symbolic procedure ")
```

The following symbols currently trigger *structure completion*:

```
("begin" . reduce-insert-block)
("ift" . reduce-expand-if-then)
("ife" . reduce-expand-if-then-else)
("<<" . reduce-insert-group)
```

They operate in exactly the same way as if the appropriate structure insertion command had been executed directly, and they receive any prefix argument entered before the completion key.

For the full set of completions and abbreviations see the customizable user option `reduce-completion-alist`.

M-TAB
M-x reduce-complete-symbol

Perform completion on the REDUCE symbol preceding point (or preceding the region if it is active). Compare that symbol against the elements of `reduce-completion-alist`. If a perfect match (only) has a `cdr` then delete the match and insert the `cdr` if it is a string or call it if it is a (nullary) function, passing on any prefix argument (in raw form).

10 Font-lock support for automatic font selection

Font-lock mode causes Emacs to select automatically the font in which text is displayed ("fontify" it) so as to indicate its logical status. See Section "Font Lock mode" in *The Emacs Editor*. The first version of font-lock support for REDUCE mode was contributed by Rainer Schoepf. The current version provides 4 levels of decoration, which can be selected using the standard font-lock facilities, or interactively most easily via the REDUCE mode font-lock sub-menu. The levels and corresponding highlighting are as follows.

1. Minimum: main keywords and group delimiters, procedure and other main type declarations, strings, %-comments
2. Algebraic: minimum plus labels, algebraic-mode declarations and variables
3. Symbolic: minimum plus labels, symbolic-mode declarations and variables
4. Maximum: all the above plus function calls and all uses of variables

Minimum highlighting is probably a good general-purpose level for normal use, but if you program almost exclusively in either the algebraic or symbolic mode of REDUCE then you might prefer the highlighting tailored to that mode. Maximum highlighting is probably too gaudy and too slow for general use; also the code is complicated and not well tested, so maximum highlighting is likely to be the least reliable.

REDUCE mode does not make any (user configurable) face definitions of its own and only standard font-lock faces are used. The faces used to highlight particular syntactic elements are as follows:

`font-lock-builtin-face`
: not currently used

`font-lock-comment-face`
: %-comments (only)

`font-lock-constant-face`
: labels

`font-lock-function-name-face`
: procedure, operator and module names *in their definition statements only* (which may change in a later version)

`font-lock-keyword-face`
: main REDUCE keywords

`font-lock-string-face`
: strings

`font-lock-type-face`
: type declaration keywords

`font-lock-variable-name-face`
: variables

`font-lock-warning-face`
: currently used only by REDUCE run mode

`font-lock-reference-face`
: obsolete and replaced by `font-lock-constant-face`

Font-lock mode can be turned on interactively in the normal way that any minor mode is turned on, e.g. it can be toggled on and off by the command `font-lock-mode`. It can also be turned on and off via the REDUCE mode font-lock sub-menu. To turn on font-lock mode automatically with REDUCE mode, put this in your `.emacs` file:

```
(add-hook 'reduce-mode-hook 'turn-on-font-lock)
```

To control the operation of font-lock mode use the customization buffer for the `Font Lock` group. The default level of fontification used by any mode can be specified by customizing the user option `font-lock-maximum-decoration`, which REDUCE mode respects.

Emacs provides standard facilities to control the use of different display faces. See Section "Using Multiple Typefaces" in *The Emacs Editor*. See Section "Faces" in *The GNU Emacs Lisp Reference Manual*, for further technical detail. To alter the appearance of a Font Lock face, use the customization buffer for the `Font Lock Highlighting Faces` group. See Section "Customizing Faces" in *The Emacs Editor*.

REDUCE mode passes information to font-lock mode via the value of the buffer-local variable `font-lock-defaults`, which could be re-set or modified via the REDUCE mode hook, although this is not recommended.

For more information see the description of the command `font-lock-mode` and related commands and variables, and/or the ELisp source code file `font-lock.el`.

Standard font-lock fontification can be slow. An elegant solution is provided by the `lazy-lock` package, which immediately fontifies only the visible part of the buffer and fontifies the remainder "stealthily" in the background when Emacs is not otherwise busy. This makes font-lock mode eminently usable even on a relatively low-powered computer (provided it has a suitable display) and I recommend it!

Font-locking of major syntactic elements, such as comments and strings, is normally controlled by the syntax table for the text being edited. This leads to a problem with a language such as REDUCE, because the character `!` represents an escape character within an identifier but not within a string. This is different from the convention in the languages (C and Emacs Lisp) that Emacs was primarily designed to support, in which the significance of the escape character does not depend on the context. I have not found a completely satisfactory way to deal with this problem. The solution currently adopted in REDUCE mode is to use a recently added font-lock facility that allows the syntax of `!` to be reset from escape to punctuation when it occurs immediately followed by a double quote, i.e. as `!"`. This avoids most of the difficulties, but it fails in the (fairly rare) case that `!"` appears in an identifier (which it does in one or two of the core REDUCE system files).

11 Access to procedures and operators

REDUCE mode can provide information about the procedure that point is currently in, and easy access via the `Imenu` facility to all the procedure and operator definitions within the current file. Whilst `Imenu` provides a convenient way to find a procedure or operator definition rapidly in the current file, the standard Emacs "tag" facility is the best way to find a procedure or operator definition rapidly in another file.

11.1 Show procedure mode

When editing or viewing long procedure definitions it is easy to forget which procedure you are looking at when the procedure statement itself is off the top of the screen. REDUCE mode can show in the mode line the name of the procedure (if any) that point is in. This facility is turned on and off by the command *`M-x reduce-show-proc-mode`* or via the REDUCE mode menu; it is off by default. (It is analogous to the standard Emacs "Which Function" mode, but it is implemented differently and largely independently.)

`M-x reduce-show-proc-mode`

> Toggle REDUCE Show Proc mode. With a prefix argument, turn REDUCE Show Proc mode on if and only if the argument is positive. When REDUCE Show Proc mode is enabled, display the current procedure name in the mode line after `reduce-show-proc-delay` seconds of Emacs idle time.

11.2 Imenu support

REDUCE mode supports the standard Emacs `Imenu` facilities (see Section "Imenu" in *The GNU Emacs Lisp Reference Manual*). The easiest way to use them is via the REDUCE menu entry that builds a new (nested) menu of REDUCE procedure and operator names. Selecting an entry in this menu moves point to the start of the definition of the specified procedure or operator. Another way to use `Imenu` is by entering the extended command *`M-x imenu`* and then using the standard Emacs completion facilities to select a procedure or operator name. The REDUCE mode `Imenu` menu-bar menu name and the regular expression used to build menu entries can be customized (see Chapter 13 [Customization of the REDUCE IDE], page 16).

11.3 Support for tag files

A REDUCE mode submenu provides rapid access to some of the main facilities for finding a definition via a tag file. Two commands (and submenu options) are also provided to facilitate tagging all the files in one directory or all the files in all subdirectories (one level deep) of a directory. The former is useful for tagging all the files associated with a single project or package; the latter for tagging all REDUCE packages in a single tag file.

Beware: Both the tagging commands currently require a UNIX-compatible command shell called `sh` in a directory in Emacs' `exec-path`. A standard Microsoft Windows (or MS-DOS) command shell *will not suffice* for this application, but there are plenty of suitable free shells that will work under Windows. I recommend the port of `ash` included in the Cygwin (`http://sourceware.cygnus.com/cygwin/`) package, which includes a port of a great deal of excellent GNU (and other) free software to Microsoft Windows. (Note that, whilst `bash` will work perfectly well, it is overkill for this application.)

M-x reduce-tagify-dir
: Generate a REDUCE TAGS file for (all `.red` files in) a directory, by default the current directory.

M-x reduce-tagify-subdirs
: Generate a REDUCE TAGS file for (all `.red` files in) all subdirectories of a directory, by default the parent of the current directory.

12 Miscellaneous minor features and bugs

REDUCE mode extends some of the standard Emacs handling of parenthesised "lists" to include REDUCE group and block constructs. It provides a major mode menu and easy access to its version information. This chapter also discusses known and potential problems using REDUCE mode, and describes how to access some special function keys that are useful in Emacs in general and in REDUCE mode in particular.

12.1 Groups and blocks: delimiter highlighting and moving over

Delimiters for groups (`<<` and `>>`) and blocks (`begin` and `end`) are treated as brackets. Either highlighting of matching group and block delimiters (consistent with the `paren` library (see Section "Automatic Display Of Matching Parentheses" in *The Emacs Editor*)) or (group only) blink matching is toggled by the command `reduce-show-delim-mode`. Highlighting is turned on automatically when REDUCE mode is selected if `reduce-show-delim-mode` is non-nil, which it is by default if the `paren` library is in use. See Section "Automatic Display Of Matching Parentheses" in *The Emacs Editor*, all of which applies to this extended bracket matching facility.

Beware: Currently highlighting of matching group and block delimiters works only when point is *at the end* of the delimiter. This may be changed in future.

M-x reduce-show-delim-mode
: Toggle REDUCE Show Delim mode. With a prefix argument, turn REDUCE Show Delim mode on if and only if the argument is positive. When REDUCE Show Delim mode is enabled, any matching delimiter is highlighted after `show-paren-delay` seconds of Emacs idle time. See Chapter 13 [Customization of the REDUCE IDE], page 16.

Groups and blocks are regarded as "symbolic expressions" (sexp's) by the commands `reduce-backward-sexp` and `reduce-forward-sexp`.

C-M-f
M-x reduce-forward-sexp
: Move forward across one balanced expression. With an argument, do it that many times.

C-M-b
M-x reduce-backward-sexp
: Move backward across one balanced expression. With an argument, do it that many times.

12.2 Major mode menu

REDUCE mode adds a major-mode menu called "REDUCE" to the menu bar, which is also available as a pop-up menu activated by the command `mouse-major-mode-menu` on the standard key *C-down-mouse-3*.

12.3 REDUCE mode version information

The version of REDUCE mode that is running is available as the value of the variable `reduce-mode-version`, which is a string that can be displayed in the echo area either by selecting the `Show Version` menu option from the REDUCE major mode menu or by running the command *M-x reduce-mode-version* (both of which also record it in the `*Messages*` buffer). If REDUCE mode is not running then an easy way to start it is to switch to a temporary buffer (e.g. by using *C-x b tmp*) and then switch it to REDUCE mode (by using *M-x reduce-mode*).

12.4 Known and potential problems

There is a problem with the way that REDUCE mode handles an exclamation mark ('`!`') followed immediately by a double quote '`"`'. See Chapter 10 [Font-lock support for automatic font selection], page 11. This should not be a problem in "normal" code, but it may upset the parsing of code that uses this character sequence within an identifier. It is caused by a limitation in the way that Emacs currently handles the syntax of the text being edited and is not easy to avoid completely! I am looking for a better resolution of this problem.

There is no guarantee that an arbitrary minor mode or other extension will be compatible with REDUCE mode, although I am not aware of any conflicts. Two minor modes that are known to be compatible with REDUCE mode are transient-mark-mode and delete-selection-mode (because I always use them and I recommend them!).

12.5 Special function keys

A number of "special function" keys are useful in Emacs in general and in REDUCE mode in particular, which are not directly accessible on all platforms. The following comments apply particularly to Microsoft Windows.

The standard key to terminate lines of indented code is `LFD` and the standard key to continue a comment is *Meta-LFD*, but `LFD` does not exist on a standard PC keyboard. Note that `LFD` can always be accessed via its ASCII code as *C-j*, or on some keyboards *Control-RET* generates `LFD`. In some situations, the keyboard can be re-programmed to provide this very convenient synonym, which is true of the better terminal emulators.

The standard key to complete a symbol is *Meta-TAB*, but Microsoft Windows uses this key combination for fast task switching.

When Emacs is able to read the keyboard directly, as when it is run under X or Microsoft Windows, `LFD` can be conveniently emulated as *Control-RET* and *Meta-TAB* as *Control-TAB*. A

good way to generate these and similar emulations is to put the following code in your `.emacs` file:

```
(define-key function-key-map [(control return)] [?\C-J])
(define-key function-key-map [(control meta return)] [?\C-\M-J])
(define-key function-key-map [(control tab)] [?\M-\t])
```

13 Customization of REDUCE IDE

REDUCE IDE supports a small amount of customization. The following edit-mode user options can be changed using the standard Emacs customization facilities. The main REDUCE customization group is called "REDUCE", under which are the two sub-groups "REDUCE Interface" and "REDUCE Format & Display" and two hooks.

REDUCE IDE font-lock support can be customized by resetting standard font-lock variables (see Chapter 10 [Font-lock support for automatic font selection], page 11).

Customization of REDUCE run mode is documented separately. See Section 14.4 [Customization of REDUCE run mode], page 19.

13.1 REDUCE mode interface customization

`reduce-imenu-generic-expression`
: Imenu support for procedure definitions and operator declarations. An alist with elements of the form `(MENU-TITLE REGEXP INDEX)` – see the documentation for `imenu-generic-expression`.

`reduce-imenu`
: Default value `nil`. If non-nil then REDUCE mode automatically calls `imenu-add-to-menubar` to add a Contents menu to the menubar.

`reduce-imenu-title`
: Default value `"Procs/Ops"`. The title to use if REDUCE mode adds a procedure/operator menu to the menubar.

`reduce-max-up-tries`
: Default value 2. Number of repeats of `reduce-forward` or `backward-statement` required to move up out of a block or group.

`reduce-completion-alist`
: Association list of REDUCE-mode completions searched by `reduce-complete-symbol`. Each key word or phrase to be simply completed should be a list containing a single string. If a perfectly matched string (only) is a non-trivial pair then the match is deleted and the `cdr` inserted if it is a string or called if it is a (nullary) function, passing on any prefix argument (in raw form).

13.2 REDUCE mode format & display customization

Format

`reduce-indentation`
: Default value 3. Depth of successive indentation in REDUCE code.

`reduce-comment-region-string`
: Default value '`%% `'. String inserted by `reduce-comment-region` or `reduce-comment-procedure` at the start of each line. Was named `reduce-comment-region` up to version 1555!

`reduce-auto-indent-mode`
: Default value `t`. If non-nil then conditionally re-indent the current line after `reduce-auto-indent-delay` seconds of Emacs idle time if the text just typed matches `reduce-auto-indent-regex`.

`reduce-auto-indent-delay`
: Default value 0.125. Time in seconds to delay before maybe re-indenting current line.

`reduce-auto-indent-regexp`
: Default value `"\\(else\\|end\\|>>\\)\\="`. Auto indent the current line if the text just typed matches this regexp. It should end with `\=`.

Display

`reduce-show-delim-mode`
: Default value is "on" if this makes sense. If non-nil then highlight matching group and block delimiters. Otherwise blink the open-group matching an inserted close-group.

`reduce-show-delim-match-face`
: Default value as for standard paren match highlighting (Show Paren mode). Face used for a matching REDUCE delimiter.

`reduce-show-delim-mismatch-face`
: Default value as for standard paren match highlighting (Show Paren mode). Face used for a mismatching REDUCE delimiter.

`reduce-show-proc-mode`
: Default value `nil`. If non-nil then display the current procedure name in the mode line after `reduce-show-proc-delay` seconds of Emacs idle time.

`reduce-show-proc-delay`
: Default value 0.125. Time in seconds to delay before showing the current procedure name.

13.3 REDUCE mode hooks

Hooks allow arbitrary customization that is not supported by the standard customization facilities. When the REDUCE mode library is loaded into Emacs, the last step of the loading process is to execute the function(s) assigned to the variable `reduce-mode-load-hook`. This hook would be appropriate for modifying global properties of REDUCE mode such as its key map.

When REDUCE mode is activated in a buffer the last step of its initialization process is to execute the function(s) assigned to the variable `reduce-mode-hook`. This hook would be appropriate for modifying properties local to the buffer.

See Chapter 2 [Installation of the REDUCE IDE], page 2, for further details.

`reduce-mode-load-hook`
: Default value `nil`. List of functions to be called when REDUCE mode is loaded, e.g. `require-reduce-run` to automatically load `reduce-run`. It can be used to customize global features of REDUCE mode such as its key map, i.e. it is a good place to put key bindings.

`reduce-mode-hook`
: Default value `nil`. List of functions to be called when REDUCE mode is entered. It can be used to customize buffer-local features of REDUCE mode, e.g. use `turn-on-font-lock` to turn on font-lock mode locally.

14 Running REDUCE in a buffer

REDUCE run mode is a subsidiary of REDUCE edit mode, in the sense that it requires REDUCE edit mode to be loaded before it is loaded (which it tries to ensure automatically). It provides an interface that allows a command-line (as opposed to a GUI) version of REDUCE to be run interactively in an Emacs buffer, with input from and output to that buffer.

In fact, it allows CSL and/or PSL REDUCE to be run simultaneously and independently in multiple buffers. You specify your preferred REDUCE version, which is then used by default, but there are also commands to start CSL or PSL REDUCE specifically. (The default preference is CSL REDUCE, mainly because it seems to be slightly more robust.)

A REDUCE source code file or an Emacs buffer containing REDUCE source code (in REDUCE mode) can be run as a completed REDUCE program in an independent buffer automatically named from its file or buffer.

REDUCE statements, procedures or general regions of code in a REDUCE mode buffer can be sent to REDUCE running in another buffer. If REDUCE is running in multiple buffers then you select which buffer to use. If you try to send REDUCE code to REDUCE but REDUCE is not running, then REDUCE run mode will (optionally) start REDUCE automatically.

14.1 Introduction to REDUCE run mode

REDUCE run mode provides the following facilities for running REDUCE:

- input editing and flexible re-execution of previous input;
- flexible browsing of all input and output;
- syntax highlighting of prompts, input and error messages;
- bracket and delimiter highlighting or matching as in REDUCE edit mode;
- easy REDUCE package loading, source file input and module compilation;
- seamless integration with source files being edited using REDUCE edit mode, including easy input of statements, procedure definitions, arbitrary regions or the whole file to REDUCE;
- optional automatic starting of REDUCE;
- menu access to many facilities;
- all other standard Emacs facilities, such as printing part or all of the interaction buffer;
- the full source code is free, and the package is documented and user customizable.

These facilities include most, if not all, of the facilities offered by any other REDUCE interface, with the notable exception of typeset quality display of mathematical output. It may be possible to support this in some future version.

REDUCE run mode is provided as a separate library so that if you prefers another REDUCE interface you do not need to install it. It is a subsidiary library to REDUCE edit mode and when loaded it hooks itself into and cooperates closely with REDUCE edit mode.

REDUCE run mode is based closely on the standard library `inf-lisp` by Olin Shivers. See Section "Running an External Lisp" in *The Emacs Editor*. Since this mode is built on top of the general command-interpreter-in-a-buffer (comint) mode, it shares a common base functionality and a common set of key bindings with all modes derived from comint mode. (Among these, shell mode is likely to be the most familiar. See Section "Interactive Inferior Shell" in *The Emacs Editor*.) This makes these modes easier to use.

For further documentation on the functionality provided by comint mode and the hooks available for customising it, see the file `comint.el`.

14.2 Installation of REDUCE run mode

It is probably best to use the semi-automatic installation procedure described earlier. See Chapter 2 [Installation of the REDUCE IDE], page 2. But if you want to install by hand, or understand the details of the installation process, then read on . . .

REDUCE run mode requires the library `reduce-mode` both when it is compiled and when it is loaded. It tries quite hard to locate this library, but normally it should be in the same directory as `reduce-run`.

Byte-compile the file `reduce-run.el`, put the result somewhere in your `load-path`, and put the following in your `.emacs` file:

```
(autoload 'run-reduce "reduce-run" "Run a REDUCE process" t)
```

If you would like to have automatic full access to the features of REDUCE run mode from REDUCE edit mode then also put the following in your `.emacs` file:

```
(add-hook 'reduce-mode-load-hook 'require-reduce-run)
```

14.3 Running PSL REDUCE on Windows

PSL REDUCE on Windows currently needs to be run explicitly from a shell and a reasonably easy way to arrange that is to create a text file called (say) `myredpsl.bat` containing the following two lines:

```
@echo off
cmd /c redpsl
```

This assumes you have added the REDUCE `bin` directory to your search path; if not then you need to use the full path for redpsl, which is probably `"C:/Program Files/Reduce/bin/redpsl.bat"`. Then customize *reduce-run-commands* (see the next section) to use the name of your `.bat` file as the command to start PSL REDUCE if it is on your search path, or use the full pathname of the file if it is not on your search path.

14.4 Customization of REDUCE run mode

REDUCE run mode provides a small amount of customization. The following user options can be changed using the standard Emacs customization facilities. The main REDUCE customization group is called "REDUCE", under which REDUCE run mode provides a sub-group "REDUCE Run" that allows the following options to be customized.

It is essential to check that *reduce-run-commands* is set correctly otherwise REDUCE will not run correctly. The settings of other user options are not critical.

`reduce-run-program` [User Option]

This variable is obsolete and has been replaced by 'reduce-run-commands' with different value syntax, as described below.

`reduce-run-commands` [User Option]

Default value `(("CSL" . "redcsl --nogui") ("PSL" . "redpsl"))`. The value of this variable is an association list of two elements, one for each of the main Lisp versions on which REDUCE is distributed, namely CSL and PSL. The order of these elements determines the order in which commands that do not specify the REDUCE version try to run REDUCE versions; see below. Each element is a "dotted pair" of strings, of which the first is either `"CSL"` or `"PSL"` and the second is the command to run that version of REDUCE using a command-line interface. This can include any required options, as illustrated above by the default value for CSL REDUCE. The command can be either the command name, as illustrated above, or it can be a full path name, which can contain spaces. (But spaces are not allowed in the command options.) If command names are used then those commands

must be on your search path. On most platforms, this should happen automatically, but on Microsoft Windows you either need to add the REDUCE `bin` directory to your search path or customize this variable to use full path names.

`reduce-run-prompt` [User Option]
Default value `"^\\([0-9]+[:*] \\)+"`. The regexp to recognise prompts in REDUCE run mode. The default works well for CSL REDUCE. This variable is used to initialize `comint-prompt-regexp` in the REDUCE run buffer.

`reduce-run-autostart` [User Option]
Default value `t`. If non-nil then all commands that require a REDUCE process will automatically start a new one if none is already running.

`reduce-run-multiple` [User Option]
Default value `t`. If non-nil then commands that explicitly start REDUCE will always start a new REDUCE process in a new distinct buffer, even if REDUCE is already running. Otherwise, they will re-use any appropriate running REDUCE process.

`reduce-run-mode-hook` [User Option]
Default value `nil`. The main hook for customising REDUCE run mode.

`reduce-run-load-hook` [User Option]
Default value `nil`. The hook run when REDUCE run mode is loaded. It is a good place to put key bindings.

`reduce-input-filter` [User Option]
Default value `"\\‘\\([ \t;$]*\\|[ \t]*.[ \t]*\\)\\’"`. What not to save on REDUCE run mode's input history. The value is a regular expression (regexp). The default matches any combination of zero or more whitespace characters and/or statement terminators, or any single character (e.g. *y* or *n*) possibly surrounded by whitespace.

`reduce-source-modes` [User Option]
Default value `(reduce-mode)`. Used to determine if a buffer contains REDUCE source code. If a file is loaded into a buffer that is in one of these major modes then it is considered to be a REDUCE source file by `reduce-input-file` and `reduce-fasl-file`. Used by these commands to determine defaults.

`reduce-packages-directory` [User Option]
Absolute pathname of the directory containing REDUCE packages, or nil. This variable should be set automatically when REDUCE run mode loads. It looks for the packages directory based on the location of default command used to run REDUCE. If the directory cannot be found then this variable will be set to nil. You can customize this variable to override the default setting.

14.5 Running, re-running and switching to REDUCE

The main commands for starting REDUCE are `run-reduce`, `run-csl-reduce` and `run-psl-reduce`. If the user option *reduce-run-multiple* is non-nil then they all start a new REDUCE process every time they are executed; successive REDUCE process buffers have successively higher numbers at the end of their names, e.g. `*CSL REDUCE*`, `*CSL REDUCE 1*`, `*CSL REDUCE 2*`, ... Otherwise they only start a new REDUCE process if necessary—if an appropriate REDUCE process is already running then they simply switch to it. Finally, these commands run the hooks from *reduce-run-mode-hook* (after the *comint-mode-hook* is run).

`run-reduce` [Command]
By default, this command starts the default REDUCE version. More precisely, it attempts to start the REDUCE version that appears first in *reduce-run-commands*, and if that fails then it attempts to start the REDUCE version that appears second in *reduce-run-commands*. It names the REDUCE process buffer with the REDUCE version that it runs, e.g. `*CSL REDUCE*` or `*PSL REDUCE*`.

If this command is executed with a prefix argument then it reads a command to run REDUCE from the minibuffer and runs that command. In this case, it names the REDUCE process buffer without any REDUCE version, e.g. `*REDUCE*`.

`run-csl-reduce` [Command]
This command starts CSL REDUCE and names the REDUCE process buffers that it creates `*CSL REDUCE*`, `*CSL REDUCE 1*`, etc.

`run-psl-reduce` [Command]
This command starts PSL REDUCE and names the REDUCE process buffers that it creates `*PSL REDUCE*`, `*PSL REDUCE 1*`, etc.

A couple of convenience commands allow a running REDUCE process to be re-started in the same buffer and switching from any buffer to a running REDUCE process buffer.

`re-run-reduce` [Command]
This command is only allowed in a REDUCE process buffer. If REDUCE is running in the current buffer then it is terminated. The command then reruns the version of REDUCE shown in the buffer name. (This will not work if you entered the command to run REDUCE interactively.)

`switch-to-reduce` [Command]
This command is primarily intended for internal use by other commands but it can be run interactively to switch to a running REDUCE process buffer. If the current buffer is an active REDUCE process buffer then the command does nothing; if there is only one active REDUCE process buffer then the command switches to it; otherwise, the command reads the name of an active REDUCE process buffer from the minibuffer with completion. This means that pressing the `tab` key will give a list of active REDUCE process buffers from which to select one. It remembers the last buffer used and offers that as the default, making it easy to switch repeatedly to the same REDUCE process buffer. This function is used by all commands described below that send REDUCE code fragments to a running REDUCE process.

If REDUCE is not running then this command will start REDUCE automatically by calling `run-reduce` if the user option *reduce-run-autostart* is non-nil (which by default it is).

14.6 Running complete REDUCE programs

The commands `reduce-run-file` and `reduce-run-buffer` run REDUCE code as an independent REDUCE program by starting the default version of REDUCE in a new process buffer named uniquely by the file or buffer containing the REDUCE code. This allows several REDUCE programs to be conveniently run simultaneously and independently.

`reduce-run-file` [Command]
This command reads a file name from the minibuffer, starts REDUCE in a new process buffer named uniquely by the file (unless it already exists) and inputs the file into REDUCE.

`reduce-run-buffer` [Command]
This command starts REDUCE in a new process buffer named uniquely by the current buffer (unless it already exists) and inputs the current buffer into REDUCE.

14.7 Inputting code fragments to REDUCE

When developing REDUCE code, it may be convenient to be able to test fragments by inputting them into REDUCE. The following commands are intended to be used in a REDUCE edit mode buffer to send code to a REDUCE process buffer. They all use `switch-to-reduce` to decide where to send the code. With a prefix argument, the commands also explicitly switch to the REDUCE process window.

`reduce-eval-last-statement` [Command]
This command sends the code between the beginning of the statement before point and point to REDUCE. The assumption is that point is at the end of a REDUCE statement, but this is not checked so it is possible to send an incomplete statement.

`reduce-eval-region` [Command]
This command sends the code in the selected region to REDUCE.

`reduce-eval-proc` [Command]
This command sends the procedure definition before or containing point to REDUCE.

14.8 Inputting, compiling and loading REDUCE files

The following commands deal with complete REDUCE files that do not necessarily constitute complete programs.

`reduce-input-file` [Command]
Read a file name from the minibuffer and input the file into a REDUCE process. It asks you whether you want the input echoed.

`reduce-fasl-file` [Command]
Read a file name from the minibuffer and compile it into a fast-loading ("fasl") file, for which it prompts for the name with the source file name as default. It asks you whether you want the input echoed.

`reduce-load-package` [Command]
Read a REDUCE package name from the minibuffer and load it into a REDUCE process. The read process uses completion based on the files in the REDUCE package directory specified by the variable *reduce-packages-directory*, which should automatically be set correctly when REDUCE run mode loads. Pressing the *tab* key should list the packages available, from which you can select in the usual way, such as by clicking on a package name with the mouse.

14.9 Run mode key bindings and menu

If the user option `reduce-run-autostart` is non-nil (which it is by default) then all commands that require a REDUCE process automatically start one if necessary. See Section 14.4 [Customization of REDUCE run mode], page 19. Where appropriate, input commands have their own history lists, and if run in REDUCE edit mode then any input file defaults to the file being edited.

The following key bindings are provided in both REDUCE edit and run modes:

C-c C-i
reduce-input-file
: Input a REDUCE source file into the REDUCE process. Echo it if `reduce-run-echo` is non-nil. (This key binding redefines its default meaning in REDUCE mode.)

C-c C-l
reduce-load-package
: Load a REDUCE package into the REDUCE process.

C-c C-f
reduce-fasl-file
: Compile a REDUCE source file to a FASL image in the REDUCE process. Echo the file if `reduce-run-echo` is non-nil.

The following key bindings are added to REDUCE edit mode:

C-x C-e
reduce-eval-last-statement-and-go
: Send the previous statement to the REDUCE process, and switch to its buffer. (This key binding follows Emacs convention.)

M-C-x
C-c C-e
reduce-eval-proc-and-go
: Send the current procedure definition to the REDUCE process, and switch to its buffer. (The *M-C-x* key binding follows Emacs convention.)

C-c C-r
reduce-eval-region-and-go
: Send the current region to the REDUCE process, and switch to its buffer.

C-c C-z
switch-to-reduce
: Switch to the REDUCE process buffer. With an argument, position the cursor at the end of the buffer.

Versions of the above "and-go" commands are also defined with names that omit the "-and-go" prefix, which do not switch to the REDUCE process buffer. These seem to be less useful and so are not currently bound to any keys.

The following key bindings, in addition to the defaults provided by comint mode, are provided in REDUCE run mode:

M-TAB
reduce-complete-symbol
: Perform completion on the REDUCE symbol preceding point (or preceding the region if it is active). Compare that symbol against the elements of `reduce-completion-alist`. If a perfect match (only) has a `cdr` then delete the match and insert the `cdr` if it is a string or call it if it is a (nullary) function, passing on any prefix argument (in raw form). (This key binding is exactly as in REDUCE mode. It is included explicitly here because it is one of the edit mode key bindings that is also particularly useful in run mode.)

The REDUCE run library provides a REDUCE run major mode menu and also adds a slightly modified version of this menu to the menu bar in REDUCE edit mode. These two menus provide appropriate access to all the above commands, and to echoing and highlighting control for REDUCE run mode.

15 Feedback: bug reports, suggestions, comments, . . .

Feedback to the author via SourceForge is welcome, including reports of errors or features that do not work well and suggestions for improvements or additional features.

If possible, please include details of the version of Emacs that you are using, the platform on which you are using it, and the version of REDUCE edit (and if relevant run) mode that you

are using. (This information is essential if you are reporting a bug.) An easy way to do this is to send me the Emacs and REDUCE IDE version strings. You can access the former from the standard Emacs `Help` menu and the latter from the REDUCE major mode menus, in all cases by selecting the `Show Version` menu option. The version strings can also be accessed by running the commands *`M-x emacs-version`*, *`M-x reduce-mode-version`* and *`M-x reduce-run-version`*. These menu options and commands will display the version string in the echo area at the bottom of the frame; it will also be recorded in the `*Messages*` buffer, from where it can easily be copied.

Command Index

A

autoload 2

B

backward-delete-char-untabify 3

D

describe-mode 1

I

indent-for-comment 7
indent-new-comment-line 7

L

load-library 2

M

mouse-major-mode-menu 15

R

re-run-reduce 21
reduce-auto-indent-mode 9
reduce-backward-procedure 5
reduce-backward-sexp 15
reduce-backward-statement 4
reduce-comment-procedure 7
reduce-comment-region 7
reduce-complete-symbol 11, 23
reduce-down-block-or-group 4
reduce-eval-last-statement 22
reduce-eval-last-statement-and-go 23
reduce-eval-proc 22
reduce-eval-proc-and-go 23
reduce-eval-region 22
reduce-eval-region-and-go 23
reduce-fasl-file 22, 23
reduce-fill-comment 7
reduce-forward-procedure 5
reduce-forward-sexp 15
reduce-forward-statement 4
reduce-indent-line 9
reduce-indent-procedure 5, 9
reduce-indent-region 9
reduce-input-file 22
reduce-insert-block 10
reduce-insert-group 10
reduce-insert-if-then 10
reduce-kill-procedure 5
reduce-kill-statement 4
reduce-load-package 22
reduce-mark-procedure 5
reduce-mode 2
reduce-narrow-to-procedure 6
reduce-reposition-window 6
reduce-run-buffer 21
reduce-run-file 21
reduce-show-delim-mode 14
reduce-show-proc-mode 13
reduce-tagify-dir 14
reduce-tagify-subdirs 14
reduce-unindent-line 10
reduce-up-block-or-group 4
reindent-then-newline-and-indent 9
run-csl-reduce 21
run-psl-reduce 21
run-reduce 21

S

switch-to-reduce 21, 23

Variable Index

A

auto-mode-alist 2

F

font-lock-builtin-face 12
font-lock-comment-face 12
font-lock-constant-face 12
font-lock-function-name-face 12
font-lock-keyword-face 12
font-lock-reference-face 12
font-lock-string-face 12
font-lock-type-face 12
font-lock-variable-name-face 12
font-lock-warning-face 12

R

reduce-auto-indent-delay 17
reduce-auto-indent-mode 17
reduce-auto-indent-regexp 17
reduce-comment-region-string 16
reduce-completion-alist 16
reduce-imenu 16
reduce-imenu-generic-expression 16
reduce-imenu-title 16
reduce-indentation 16
reduce-input-filter 20
reduce-max-up-tries 16
reduce-mode-hook 17
reduce-mode-load-hook 17
reduce-packages-directory 20

reduce-run-autostart 20
reduce-run-commands 19
reduce-run-load-hook 20
reduce-run-mode-hook 20
reduce-run-multiple 20
reduce-run-program 19
reduce-run-prompt 20
reduce-show-delim-match-face 17
reduce-show-delim-mismatch-face 17
reduce-show-delim-mode 17
reduce-show-proc-delay 17
reduce-show-proc-mode 17
reduce-source-modes 20

Keystroke Index

B

BACKTAB 10

C

C-c : 7
C-c ; 7
C-c < 10
C-c b 10
C-c C-d 4
C-c C-e 23
C-c C-f 23
C-c C-i 22
C-c C-k 4
C-c C-l 22
C-c C-n 4
C-c C-p 4
C-c C-r 23
C-c C-u 4
C-c C-z 23
C-c i 10
C-c k 5
C-c TAB 10, 22
C-down-mouse-3 15
C-h m 1
C-M-\ 9
C-M-a 5
C-M-b 15
C-M-e 5
C-M-f 15
C-M-h 5
C-M-l 6
C-M-q 5, 9
C-x C-e 23
C-x n d 6

D

DEL 3

L

LFD 9, 15

M

M-; 7
M-C-x 23
M-LFD 7
M-q 7
M-TAB 11, 23
Meta-LFD 15
Meta-TAB 15

S

Shift-TAB 10

T

TAB 9, 10, 15

Concept Index

A

Abbreviations 11
Access to procedures and operators 13

B

Blocks 14
Bug reports 23
Bugs 14, 15

C

Comment support 6
Comments 23
Compatibility 15
Completion 11
Contact 23
Customization 16, 19

D

Delimiters 14
Display customization 16
Displaying current procedure 13

E

Expansion 11

F

Features 2, 14
Feedback 23
Font selection 11
Font-lock support 11
Format & display customization 16
Format customization 16
Function keys 15

G

General features 2
Groups 14

H

Highlighting and moving over 14
Highlighting of keywords 11
Hooks 17

I

Imenu support 13
Indentation 8
Information about version 15
Information, procedures and operators 13
Installation of REDUCE IDE 2
Installation of REDUCE run mode 19
Interface customization 16
Introduction to REDUCE IDE 1
Introduction to REDUCE run mode 18

K

Key bindings 22
Keys, special function 15
Keyword completion 11
Keyword highlighting 11

L

LFD 15
Linefeed 15

M

Major mode menu 15
Menu 22
Menu of procedures and operators 13
Menu, Major mode 15
Meta-LFD 15
Meta-TAB 15
Minor modes 14
Miscellaneous 14
Moving over 14
Multiple Process Support 20

O

Operations on Procedures 5
Operations on Statements 3
Operator menu 13
Operators access 13
Options 16, 19

P

Problems 15
Procedure access 13
Procedure display 13
Procedure menu 13
Procedures 5
Process Support, Multiple 20
PSL on Windows 19

R

REDUCE mode version information 15
Run mode 18
Running PSL REDUCE on Windows 19
Running REDUCE 18

S

Show procedure mode 13
Showing current procedure 13
Special function keys 15
Statements 3
Structure templates 10
Structures 11
Suggestions 23
Support for imenu 13
Support for Multiple Process 20
Support for tag files 14

T

TAB 15
Tag files 14
Tags 14
Templates for structures 10

U

User options 16, 19

V

Variables 16, 19
Version information 15

W

Which procedure 13