GNU Emacs REDUCE IDE

An Integrated Development Environment for REDUCE:
Major modes for editing and running REDUCE source code
Software version 1.10

Francis J. Wright (`https://sites.google.com/site/fjwcentaur`)

Manual last updated Time-stamp: <2022-12-14 15:40:10 franc>

This manual is for REDUCE IDE (version 1.10, updated Time-stamp: <2022-12-14 15:40:10 franc>), which provides GNU Emacs major modes for editing and running REDUCE source code.

Table of Contents

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 (REDUCE mode) and a subsidiary major mode for running REDUCE as an inferior process with input and output via a buffer (REDUCE 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 written in Lisp, as is (most of) Emacs. However, the REDUCE user language is similar to Algol 60 (`https://en.wikipedia.org/wiki/ALGOL_60`), an ancestor of most current programming languages.

I began development of REDUCE mode tentatively in late 1992 and seriously in early 1994, and I began development of REDUCE Run mode in late 1998. I have continued development sporadically. Comments, suggestions, bug reports, etc. are welcome; Chapter 15 [Feedback], page 25.

REDUCE IDE is released as an Emacs package; Chapter 2 [Installation of REDUCE IDE], page 1. The latest development source code (which may not work!) is available from GitHub (`https://github.com/fjwright/REDUCE-IDE`), where package release versions are tagged (e.g. `v1.7`) and are available as releases (e.g. `reduce-ide-1.7.tar`). The source code (only) for the latest package release is also available from SourceForge (`https://sourceforge.net/p/reduce-algebra/code/HEAD/tree/trunk/generic/emacs/`).

The current version of REDUCE IDE is intended for use with GNU Emacs version 28 or later, which I will endeavour to support under recent versions of Microsoft Windows and Linux. It should also run under closely related versions of GNU Emacs and on other platforms, but I may not be able to provide support.

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

The purpose of REDUCE 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; 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 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. REDUCE Run mode is described in its own chapter; Chapter 14 [Run], page 18.

All REDUCE IDE commands are self-documenting as usual in Emacs, including in particular the modes themselves. Hence, for an overview of REDUCE mode, 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

I recommend that you use the GNU Emacs package manager to install the latest REDUCE IDE package as described in the installation section of the main REDUCE IDE web page (`https://reduce-algebra.sourceforge.io/reduce-ide/#installation`). Alternatively, releases as Emacs package (`.tar`) files are available from GitHub (`https://github.`

com/fjwright/REDUCE-IDE/releases) and via the REDUCE IDE home page (https://reduce-algebra.sourceforge.io/reduce-ide/). You can download a package file to any convenient directory and run the Emacs command *M-x package-install-file* on it. For further details, Section "Packages" in emacs. None of the manual installation described below is then required.

The rest of this chapter, and the related section of the chapter on REDUCE Run mode, are for users who want to install and configure REDUCE IDE "by hand", or who want to understand the installation process.

REDUCE mode is provided by files called reduce-mode.el, reduce-delim.el and reduce-font-lock.el, which are files of Emacs Lisp source code. These files should be byte-compiled, and the compiled (.elc) files installed in a directory from which Emacs loads its Lisp code. If necessary, you can customize your Emacs load-path so that Emacs can find the .elc files; 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 operating system 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 you can also edit it 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. (It will then normally load reduce-delim.elc and reduce-font-lock.elc automatically.) 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 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 you select 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))
```

You can use other extensions 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; Chapter 14 [Running REDUCE in a buffer], page 18.

3 General features of REDUCE mode

REDUCE mode can be selected by giving the command *M-x reduce-mode*, although normally it will be selected automatically, probably via the filename extension; Chapter 2 [Installation of the REDUCE IDE], page 1. REDUCE mode inherits from prog-mode and so shares some basic functionality common to all Emacs programming modes.

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,

they also ignore comments; Chapter 6 [Support for REDUCE comments], page 7. 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 the %-comment conventions basically follow those of Lisp mode. I have also taken some ideas from FORTRAN mode.

The standard Emacs syntax tables are modified to reflect REDUCE syntax so that, for example, Emacs knows that the REDUCE escape character is `!`.

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 `prog-mode-hook` and then 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 1. See Chapter 13 [Customization of 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!

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 forwards and backwards by statements or expressions 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 REDUCE mode looks for the beginning or end of a statement it will skip complete compound or block statements (`begin ... end`), group statements (`<< ... >>`), and bracketed expressions (`(...)`, `{...}` and `[...]`, although 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.

In particular, the REDUCE-mode movement commands may fail if point is within a comment or string, although they should skip complete comments and strings.

REDUCE mode considers REDUCE statements to be terminated by either of the characters `;` or `$`. It also considers statements contained within any kind of brackets to be delimited

by those brackets, statements within compound or block statements (`begin ... end`) to be delimited by the `begin` and `end` keywords, and statements within group statements (`<< ... >>`) to be delimited by the `<<` and `>>` tokens. Commas are not considered to delimited statements.

More precisely, a statement is considered to begin at the first non-white-space character following the previous statement terminator, opening bracket, `begin` or `<<`. It is considered to end immediately after the first statement terminator or immediately after the last non-white-space character preceding a closing bracket, `end` or `>>`. Comments are treated as white space by all REDUCE-mode commands other than those specifically related to comments; Chapter 6 [Support for REDUCE comments], page 7.

The current philosophy of REDUCE mode is that the statements within a compound or group statement form an essentially isolated system, and that the basic statement-oriented commands should not move point either into or out of this system, for which separate commands are provided. 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 option `reduce-max-escape-tries`, which currently has the default value 2; Chapter 13 [Customization of REDUCE IDE], page 16. 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.

The following commands all accept a numerical argument, which defaults to 1. The commands to move forwards or backwards by statements do not move in the opposite direction if given a negative argument, in which case they do not move at all. They contain special code to handle the keyword `end` when it is used as the end-of-file marker and they do not regard comment statements as statements, i.e. they treat them as white space. They report a user error if they fail.

C-c C-n
M-x reduce-forward-statement
: Move forwards to the end of the current statement if within a statement or to the end of the following statement otherwise. 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-escape-tries` consecutive interactive tries.

C-c C-p
M-x reduce-backward-statement
: Move backwards to the start of the current statement if within a statement or to the start of the previous statement otherwise. With an argument, do it that many times. If looking at the beginning of a block or group move over it after `reduce-max-escape-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 from point. With a prefix argument, kill that many statements from point. Negative arguments kill statements backwards, where the prefix argument minus (-) is equivalent to -1.

C-c C-u
M-x reduce-up-block-or-group
: Move backwards up one level of block or group, i.e. to the beginning of the `begin` or `<<` at the start of the block or group containing point. A universal argument means move forwards, i.e. to the end of the `end` or `>>` at the end of the block or group containing point. Report a user error if the move fails. With a numeric argument,

do it that many times, where a negative argument means move forwards instead of backwards.

C-c C-d
M-x reduce-down-block-or-group
Move forwards down one level of block or group, i.e. to the end of the nearest `begin` or `<<` within the current block or group, if any. A universal argument means move backwards to the beginning of the nearest `end` or `>>` within the current block or group, if any. Report a user error if the move fails. With a numeric argument, do it that many times, where a negative argument means move backwards instead of forwards.

The following two commands move by "balanced expression", which means a symbol, string, bracketed expression, block or group. A symbol or bracketed expression may be quoted. The commands skip any preceding or intervening white space or terminator characters, but assume point is not in a string or comment.

C-M-f
M-x reduce-forward-sexp
Move forwards across one "balanced expression". With a numeric argument, move that many times, where a negative argument means move backwards instead of forwards. This command is modelled on `forward-sexp`.

C-M-b
M-x reduce-backward-sexp
Move backwards across one "balanced expression". With a numeric argument, move that many times, where a negative argument means move forwards instead of backwards. This command is modelled on `backward-sexp`.

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 one of the procedural keywords `procedure`, `listproc` or `matrixproc` within their header. Procedure type declarations (`symbolic`, `inline`, `real`, etc.) preceding the procedural keyword are also supported.

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 and kill a procedure also include *all* blank lines before the procedure definition. Many 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 forwards to the end of the procedure ending after point. With a positive argument, do it that many times. If this fails, move forwards by as many complete procedures as possible and report a user error. Skip to the first following non-blank character or the next line.

C-M-a
M-x reduce-backward-procedure
: Move backwards to the start of the procedure starting before point. With a positive argument, do it that many times. If this fails, move backwards by as many complete procedures as possible and report a user error. Skip to the start of any procedural types.

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. One way to move to the start of the next procedure is to move forwards to its end and then move backwards to its start.

Marking is the basis for many operations on procedures.

C-M-h
M-x reduce-mark-procedure
: Mark the procedure ending after point. With a positive argument, mark that many procedures ending after point. Put mark at the first non-blank character or next line after the appropriate end of procedure. If this fails, do not mark anything and report a user error. Leave point at the start of the first procedure before any preceding blank lines.

C-c k
M-x reduce-kill-procedure
: Kill the procedure ending after point. With a positive argument, kill that many procedures ending after point, including any preceding blank lines. If this fails, do not kill anything and report a user error.

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
: Narrow to the procedure ending after point. In other words, make all text outside this procedure invisible. With a positive argument, include that many procedures ending after point. Also include any preceding blank lines. If narrowing fails, report a user error. See Section "Narrowing" in *The Emacs Editor*.

6 Support for REDUCE comments

There are three 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 terminator. 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. C-style comments, which begin with `/*` and end with `*/`, are also accepted although not (yet) widely used. REDUCE mode highlights them all as comments.

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 `%` and `/**/` comments can contain statement terminators that do not have any syntactic significance.) There is currently no way to use any of the REDUCE syntax-directed commands on comment statements.

The REDUCE mode indentation and fill commands support all three comment types, but there is no other support for comment statements or `/**/` comments. 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 `comment-dwim`, normally bound to the key *`M-;`*. An existing `%`-comment can be automatically continued on the next line by the standard Emacs command `default-indent-new-line`, normally bound to the keys *`M-j`* and *`C-M-j`*. 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. *`S-TAB`*) keys. See Chapter 7 [Indenting REDUCE code automatically], page 8.

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

`C-M-j`

`M-j`
`M-x default-indent-new-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.

M-q
M-x reduce-fill-comment
: Fill a comment statement, successive `%`-comment lines or a `/**/` comment 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. toggling 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 spaces determined by the value of the variable `reduce-indentation`, the default value of which is 3; Chapter 13 [Customization of REDUCE IDE], page 16. 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 inserted 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 may seem 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 *C-j* runs the command `reindent-then-newline-and-indent` rather than just `newline-and-indent`.

The command to indent, or re-indent, a line of text is `reduce-indent-line`, normally bound (indirectly) to the key `TAB`. 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 contains only white space. Normally, however, the most convenient way to use automatic indentation is to terminate each line of code with *C-j* rather than `RET`. See Chapter 12 [Miscellaneous minor features], page 14.

When called with any argument, which is possible only via the direct key binding *M-i*, `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-i
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. With an interactive argument, indent any additional lines of the same statement rigidly together with this one.

C-j
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 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 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. 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.

The command `reduce-indent-line-always`, bound to *C-TAB*, is analogous to `reduce-indent-line` but always indents by precisely one additional step consisting of `reduce-indentation` spaces. With an argument, it rigidly indents by one step the current line and the rest of the logical statement.

The command `reduce-unindent-line`, bound to *S-TAB* (i.e. `BACKTAB`), is an inverse of `reduce-indent-line-always` and decreases the indentation by one step. With an argument, it rigidly unindents by one step the current line and the rest of the logical statement.

C-TAB
M-x reduce-indent-line-always
: Indent the current line as REDUCE code by adding `reduce-indentation` spaces at the beginning of the line. With an interactive argument, indent any additional lines of the same statement rigidly along with this one.

BACKTAB

S-TAB
M-x reduce-unindent-line
: Unindent the current line as REDUCE code by deleting `reduce-indentation` spaces (or as many as possible) 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 Chapter 12 [Miscellaneous minor features and bugs], page 14.) This works in a similar way to completion in other major modes; 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 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 Schöpf. The current version provides 3 strictly inclusive level: "Symbolic" includes "Algebraic" includes "Basic". The level can be selected using the standard font-lock facilities, or interactively most easily via the REDUCE mode *Syntax Highlighting* sub-menu. The levels and corresponding highlighting are as follows:

0. Basic: strings are displayed using `font-lock-string-face`; all types of comment (comment statements, comments from `%` to the end of the line, and C-style comments between `/*` and `*/` delimiters) are displayed using `font-lock-comment-face`; the main keywords including block delimiters and group delimiters are displayed using `font-lock-keyword-face`.
1. Algebraic: as basic level; additionally procedure names are displayed using `font-lock-function-name-face` and procedure parameters are displayed using `font-lock-variable-face`; general types such as "algebraic" are displayed using `font-lock-type-face`; local variable types such as "scalar" are displayed using `font-lock-type-face` and variable names declared local are displayed using `font-lock-variable-name-face`; goto and label names and named constants such as "pi" and "Catalan" are displayed using `font-lock-constant-face`; declared operator, operator type such as "linear", vector, array and matrix declarations are displayed using `font-lock-type-face`; operator, vector, and array and matrix identifiers (with or without bounds) are displayed using `font-lock-function-face`.
2. Symbolic: as algebraic level; additionally preprocessor #-directives are displayed using `font-lock-preprocessor-face`; the key symbolic-mode "types" `fluid` and `global` are displayed using `font-lock-type-face` and the variable names declared fluid or global are displayed using `font-lock-variable-name-face`; all other quoted objects are displayed using `font-lock-constant-face`; asserted types are displayed using `font-lock-type-face`; "declare" and "struct" statements, as used in "redlog", are displayed using `font-lock-keyword-face`, `font-lock-function-name-face` and `font-lock-type-face`; module, endmodule and other symbolic-mode keywords are displayed using `font-lock-keyword-face`; module names are displayed using `font-lock-constant-face`; key symbolic-mode functions such as "flag" and "get" are displayed using `font-lock-builtin-face`; other symbolic-mode "types" such as "switch" and "share" are displayed using `font-lock-type-face`; declared lambda parameters are displayed using `font-lock-variable-face`.

Using Emacs' default settings gives symbolic-level highlighting, but if you find this too gaudy or too slow then you might prefer to select a lower level. 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 *Syntax Highlighting* 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, customize the appropriate options in the `Font Lock` group. The default level of fontification used by any mode can be specified by customizing the 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`.

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. The solution I adopt in REDUCE mode is to reset the syntax of `!` from escape to punctuation when it occurs immediately followed by a double quote, i.e. as `!"`, but only within a string.

11 Access to procedure and operator definitions

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 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 procedure definition via a tag file. Two commands (and submenu options) facilitate tagging the REDUCE files in one directory or in a directory and all its sub-directories. 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.

Once a TAGS file has been generated, the standard Emacs `xref` interface is available to find identifier references, in particular the command `xref-find-definitions`, which is also available via the REDUCE mode submenu. See Section "Find Identifier References" in *GNU Emacs Manual*.

The tagging commands use the standard Emacs `etags` program, which should be available in the Emacs `bin` directory. REDUCE mode uses the value of the option `reduce-etags-directory`, which should be appropriate by default, but if not can be customized. (It is not required that the Emacs `bin` directory be in your execution path, but if it is you can optionally set `reduce-etags-directory` to nil.)

M-.

M-x xref-find-definitions
: Show the definitions of the identifier at point. With a prefix argument, or if there's no identifier at point, prompt for the identifier.

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

M-x reduce-tagify-dir-recursively
: Generate a REDUCE TAGS file for (all `.red` files in) the specified directory, by default the current directory, and all its subdirectories. The single TAGS file goes in the specified directory.

12 Miscellaneous minor features

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 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

Delimiters for groups (`<<` and `>>`) and blocks (`begin` and `end`) are treated like brackets. Either highlighting of matching group and block delimiters or (group only) blink matching is toggled by the command `reduce-show-delim-mode`.

REDUCE Show Delim mode is based closely on Show Paren mode; Section "Automatic Display Of Matching Parentheses" in *The Emacs Editor*. It is completely independent except that all settings default to the corresponding values for Show Paren mode. Show Delim mode is turned on automatically when REDUCE mode is selected if `reduce-show-delim-mode-on` is non-nil, which it is by default if Show Paren mode is on.

Note that highlighting of matching group and block delimiters does not work when point is *within* a delimiter. (This cannot happen with brackets, which are single characters). 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 `reduce-show-delim-delay` seconds of Emacs idle time. See Chapter 13 [Customization of REDUCE IDE], page 16.

Show Delim mode is a buffer-local minor mode. Whenever point is before an opening delimiter or after a closing delimiter, the delimiter, its matching delimiter, and optionally the text between them are highlighted. To customize REDUCE Show Delim mode, type *M-x customize-group RET reduce-delim-showing*. The customizable options which control the operation of this mode include the following:

- `reduce-show-delim-highlight-opendelim` controls whether to highlight an opening delimiter when point stands just before it, and hence its position is marked by the cursor anyway. The default is the value of `show-paren-highlight-openparen`.
- `reduce-show-delim-style` controls whether just the two delimiters, or also the space between them get highlighted. The valid options here are `delimiter` (show the matching delimiter), `expression` (highlight the entire expression enclosed by the delimiters), and `mixed` (highlight the matching delimiter if it is visible, the expression otherwise). The default is determined by the value of `show-paren-style`.
- `reduce-show-delim-when-point-inside-delim`, when non-`nil`, causes highlighting also when point is immediately inside a delimiter. The default is the value of `show-paren-when-point-inside-paren`.
- `reduce-show-delim-when-point-in-periphery`, when non-`nil`, causes highlighting also when point is in whitespace at the beginning or end of a line, and there is respectively an opening or closing delimiter as the first or last non-whitespace characters on the line. The default is the value of `show-paren-when-point-in-periphery`.

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*.

REDUCE Run mode adds a second major-mode menu called "Run REDUCE", which appears immediately to the right of the "REDUCE" menu on the menu bar and immediately below the "REDUCE" menu on the pop-up menu.

When REDUCE Run mode is not loaded, it is possible to have an abbreviated "Run REDUCE" menu instead of the full menu. The abbreviated menu can only run REDUCE (which implicitly loads REDUCE Run mode) or explicitly load REDUCE Run mode. Whether or not to load REDUCE Run mode or install the abbreviated REDUCE Run mode menu when REDUCE mode loads is controlled by the option `autoload-reduce-run`. See Section 14.9 [Run Customization], page 23.

12.3 REDUCE IDE version information

The version of REDUCE IDE that is running is available as the value of the variable `reduce-ide-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-ide-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 Special function keys

The standard key to complete a symbol is *Meta-TAB*, but Microsoft Windows uses this key combination (referred to as `Alt + Tab`) to switch between open apps. However, `TAB` can always be accessed via its ASCII code as *C-i*, so *Meta-TAB* can always be accessed as *C-M-i*.

13 Customization of REDUCE IDE

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

REDUCE mode inherits from `prog-mode`, so some of its options also affects this mode. The `prog-mode` customization group can be accessed via a link in the REDUCE customization group.

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 12).

Customization of REDUCE Run mode is documented separately. See Section 14.9 [Customization of REDUCE Run mode], page 23.

13.1 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 1, for further details.

`prog-mode-hook`
: Default value `nil`. Normal hook run when entering programming modes. Functions added to this hook apply to all programming modes. They are run *before* any functions on `reduce-mode-hook`.

`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.

`reduce-mode-load-hook`
: Default value `nil`. List of functions to be called when REDUCE mode is loaded. 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.

  Adding the function `require-reduce-run` to this hook to automatically load `reduce-run` is now deprecated and the function will be removed in the next release; instead, please customize the new option `autoload-reduce-run`. See Section 14.9 [Run Customization], page 23.

13.2 REDUCE mode display customization

Options that control how REDUCE source code is displayed in REDUCE mode but have no effect on the code itself.

`reduce-font-lock-mode-on`
: Default value `t`. If non-nil then turn on `reduce-font-lock-mode` initially. More

`reduce-show-delim-mode-on`
: Default value the value of `show-paren-mode`. If non-nil then turn on `reduce-show-delim-mode` initially in each REDUCE mode buffer.

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

`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.

13.2.1 Subgroup: Reduce Delim Showing

Further details of showing (un)matching of group/block delimiters and enclosed expressions.

13.3 REDUCE mode format customization

Options that control the automatic formatting of REDUCE source code by REDUCE mode editing commands.

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

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

`reduce-comment-region-string`
: Default value '`%%` '. String inserted by `reduce-comment-region` or `reduce-comment-procedure` at the start of each line.

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

13.4 REDUCE mode interface customization

Options that control the REDUCE mode user interface.

`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).

`reduce-etags-directory`
: Directory containing the `etags` program, or `nil` if it is in path. If non-nil the string must end with `/`.

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

`reduce-max-escape-tries`
: Default value 2. Number of repeats of `reduce-forward-statement` or `reduce-backward-statement` required to escape out of a block or group. (This option was previously called `reduce-max-up-tries`.)

14 Running REDUCE in an Emacs window

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

In fact, it allows you to run multiple versions of REDUCE simultaneously and independently in multiple windows. By default, it runs the distributed pre-built versions of CSL and PSL REDUCE.

You can also run a REDUCE source code file or an Emacs buffer containing REDUCE source code (in REDUCE mode) as a complete REDUCE program in an independent window automatically named from the source code 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 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.

These facilities include many of those offered by other REDUCE interfaces, with the notable exception of typeset quality display of mathematical output.

REDUCE Run mode is a subsidiary library to REDUCE mode and when loaded it hooks itself into and cooperates closely with REDUCE mode.

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

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

Note that, by default, in comint mode the whole interaction buffer is editable, which applies also to REDUCE Run mode, unlike most other REDUCE interfaces, in which previous input and output cannot be changed. This can be particularly disconcerting when deleting erroneous input "back to the prompt" because, by default, the prompt itself can also be deleted. This can be prevented by customizing the option `comint-prompt-read-only`. The `comint` customization group can be accessed via a link in the REDUCE Run customization group.

14.2 Installation of REDUCE Run mode

It is probably best to use the GNU Emacs package manager to install the latest REDUCE IDE package; Chapter 2 [Installation of REDUCE IDE], page 1. None of the manual installation described below is then required. 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)
```

To control access to the features of REDUCE Run mode from REDUCE mode you can customize the option `autoload-reduce-run`. See Section 14.9 [Run Customization], page 23.

14.3 Running REDUCE on Microsoft Windows

On Microsoft Windows, the REDUCE installer does not add the REDUCE executable folder to your path, so you can run REDUCE only via absolute pathnames, which is what the shortcuts added to the Start menu do. Therefore, REDUCE Run mode also uses absolute pathnames if it can find them, which it does by looking for the installation directory `\Program Files\Reduce\` in all local drives. It uses relative batch file names if it fails to determine the absolute pathnames, but this will work only if you have added the REDUCE executable folder to your path yourself.

If PSL REDUCE on Windows is run directly via the standard batch file `redpsl.bat` it is unable to load any packages (or modules). The workaround that I use is to run PSL REDUCE indirectly via a batch file called `reduce-run-redpsl.bat`, which then runs `redpsl.bat`. (I don't understand why this works, but it does!) When `reduce-run` is loaded, it attempts to create `reduce-run-redpsl.bat` in the same folder as `reduce-run`, and if that is successful then it automatically configures the default value of `reduce-run-commands` to use `reduce-run-redpsl.bat`

to run PSL REDUCE. Provided this all works, PSL REDUCE should run as you would expect. There is no problem running CSL REDUCE via the standard batch file `redcsl.bat`.

You can customize the values of the options `reduce-run-MSWin-drives`, `reduce-run-installation-directory` and `reduce-run-commands` (see the next section) if the default values are not correct. For example, if REDUCE is installed on a remote drive then make that the sole entry in `reduce-run-MSWin-drives`.

14.4 Running, re-running and switching to REDUCE

The command to start REDUCE is `run-reduce`. If the option `reduce-run-multiple` is non-nil (which by default it is) then it starts a new REDUCE process every time it is 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 it only starts a new REDUCE process if necessary – if an appropriate REDUCE process is already running then it simply switches to it. Finally, these commands run the hooks from `reduce-run-mode-hook` (after the `comint-mode-hook` is run).

`run-reduce` [Command]

This command prompts for the name of a REDUCE command in the minibuffer, using completion and ignoring case. The REDUCE command names are as specified by the option `reduce-run-commands`. If you don't give a command name, i.e. you just hit `RET`, then a menu pops up at the current pointer location, from which you can select the name of the command you want to run. The REDUCE process buffer is named using the name of the REDUCE command that it runs, e.g. `*CSL REDUCE*` or `*PSL REDUCE*`.

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

A couple of convenience commands allow you to re-start a running REDUCE process in the same buffer and switch 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 REDUCE command whose name is shown in the buffer name. (This will not work if you entered the actual command to run REDUCE directly.)

`switch-to-reduce` [Command]

This command is primarily intended for internal use by other commands but it can be used 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 option `reduce-run-autostart` is non-nil (which by default it is).

14.5 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.

14.6 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 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.

14.7 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.

14.8 Run mode key bindings and menu

Pressing `RET` at the end of a line of input sends it to REDUCE, as with the standard REDUCE interfaces. However, if there is no terminator at the end of the line then a `;` is automatically appended, unless there is a `?` in the input line (implying that the input is the response to a user query, which must not be followed by a terminator). You can avoid this automatic behaviour completely by holding down the `SHIFT` key, i.e. by pressing `SHIFT-RET`.

RET
reduce-run-send-input
: Send the preceding input to REDUCE after appending a `;` if appropriate; holding down the `SHIFT` key avoid this.

If the 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 14.9 [Customization of REDUCE Run mode], page 23. Where appropriate, input commands have their own history lists, and if run in REDUCE 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 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 REDUCE mode key bindings that is also particularly useful in REDUCE 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 mode. These two menus provide appropriate access to all the above commands, and to echoing and highlighting control for REDUCE Run mode.

14.9 Customization of REDUCE Run mode

REDUCE Run mode provides a small amount of customization and the following 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.

The option `reduce-run-commands` must be set correctly otherwise REDUCE will not run correctly. The settings of other options are less critical.

REDUCE Run inherits from comint, so comint options also affects this mode. The `comint` customization group can be accessed via a link in the REDUCE Run customization group.

`autoload-reduce-run`
: Whether, and if so how, to autoload REDUCE Run mode. Loading it is necessary only if you plan to run REDUCE within REDUCE IDE. If the value is `t` then load REDUCE Run mode after `reduce-mode` has loaded; if it is `menu` (the default) then display a Run REDUCE menu stub that can load REDUCE Run mode; if it is `nil` then do nothing.

`reduce-input-filter`
: 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-packages-directory`
: Absolute pathname of the directory containing REDUCE packages, or nil. This option should be set automatically when REDUCE Run mode loads. If the directory cannot be found then this option will be set to nil. You can customize this variable to override the default setting.

`reduce-run-mswin-drives`
: On Microsoft Windows, this is the list of drives to be searched for REDUCE, which defaults to all local drives, e.g. `("C:" "D:" "E:" "F:")`. It is undefined on other platforms and is used only to set the default value of `reduce-run-installation-directory`.

`reduce-run-autostart`
: 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-commands`
: Default value `(("CSL" . "redcsl --nogui") ("PSL" . "redpsl"))` except on Microsoft Windows. The value of this variable is an association list with one element for each version of REDUCE, by default CSL and PSL. Each element is a "dotted pair" of strings, of which the first defaults to 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, which should be the case automatically on platforms other than Microsoft Windows.

  On Microsoft Windows, REDUCE Run mode uses full pathnames by default and moreover it uses a separate batch file to run PSL REDUCE, as described above (see Section 14.3 [REDUCE on Windows], page 19).

`reduce-run-installation-directory`
: Default value `X:/Program Files/Reduce/` on Microsoft Windows, where X is a letter A-Z, and `/usr/share/reduce/` on other platforms. Absolute root directory of the REDUCE installation, or nil if not set. It is the directory containing the `packages` directory and, on Microsoft Windows, the `bin` directory containing the user-executable batch files. On Microsoft Windows, REDUCE Run mode attempts to determine the correct value for this variable automatically. Note that you can complete the directory name using *M-TAB*.

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

`reduce-run-mode-hook`
: Default value `nil`. The main hook for customizing REDUCE Run mode.

`reduce-run-multiple`
: 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-prompt`
: Default value `"^\\(?:[0-9]+[:*] \\)+"`. The regexp to recognise prompts in REDUCE Run mode. This variable is used to initialize `comint-prompt-regexp` in the REDUCE run buffer.

`reduce-source-modes`
: 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.

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

Feedback is welcome, including reports of errors or features that do not work well and suggestions for improvements or additional features. You can provide feedback in several ways:

- informally using email (or a web form) via my website (`https://sites.google.com/site/fjwcentaur/feedback`);
- via my GitHub issue tracker (`https://github.com/fjwright/REDUCE-IDE/issues`), for which you must be logged in to GitHub;
- via the main REDUCE bug tracker (`https://sourceforge.net/p/reduce-algebra/bugs/`), for which you must be logged in to SourceForge.

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 IDE 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`* and *`M-x reduce-ide-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

C

D

L

M

R

S

Variable Index

A

P

R

Keystroke Index

B

C

M

R

S

T

Concept Index

A

B

C

D

E

F

G

H

I

K

M

O

P

R

S

T

V

W