The Implementation of WHYSE

Bryce Carson

August 10, 2024

Contents

1 WHYSE Projects 1
- 1.1 Widgetry . . . 2
- 1.2 Database initialization . . . 5
- 1.3 Customizing the behaviour of `whyse` with hooks . . . 6

2 Parsing project nowebs 7
- 2.1 PEG rules . . . 7
  - 2.1.1 indexing . . . 11
  - 2.1.2 cross referencing . . . 13

3 Processing parsed nowebs into SQL 16
- 3.1 Processing . . . 16
  - 3.1.1 Reducing complexity in the alist . . . 17
  - 3.1.2 finding the section title of a chunk . . . 20
  - 3.1.3 Determining parent-child relationships . . . 22

A Packaging whyse 23

B TESTING 25
- B.1 Parsing tool syntax within a temporary buffer . . . 25

C Indices 26
- C.1 Chunks . . . 26
- C.2 Identifiers . . . 27
- C.3 Miscellaneous code and functions useful for development and debugging . . . 28

D License 30

Abstract

(no)Web HYpertext System in Emacs (WHYSE) is an integrated development environment for Noweb and LaTeX within Emacs, similar to EDE but not sharing development principles. It is based off of CITE an academic paper written in 1991 by Brown and Czejdo. A paper describing this implementation—written in Noweb and browsable, editable, and auditable with WHYSE, or readable in the printed form—is hoped to be submitted to The Journal of Open Source Software (JOSS) before the year 2024. N.B.: the paper will include historical information about literate programming, and citations (especially of those given credit in the `<<Commentary>>` for ideating WHYSE itself).

Users of WHYSE in Emacs are expected to be familiar with Noweb; this does not include how Noweb is built from source (that is arcane, supposedly). It may, however, include the writing of filters implemented with Sed, AWK, or other languages. Users must know how to write a custom command-line for noweave (read the manual section regarding the `-v` option). If you only know how to call the noweave command you're reading the wrong document. Read the Noweb manual first, please. Developers of WHYSE extensions should read the Noweb Hacker's Guide until they understand it, afterwards reading this documentation several times until the full implementation is understood. I recommend modifying the system using itself to keep organized, and writing literately; you'll thank yourself later for doing so.

Chapter 1

WHYSE Projects

The organization of this literate program is linear, with aspects of the program explained as the user would encounter them, more or less. A user will read from the package description that they should call an interactive command to create a project. The WHYSE application has a single interactive command: `whyse`. The command loads the first element of the customization variable `w-registered-projects`, considering that the default project, or it opens the "Easy Customization Interface" for the application's customization group (`M-x customize-group whyse`): an effective prompt for the user to enter the necessary information. If user's dislike this, they can disable it.

A customization group for WHYSE is defined to organize its customization variables, and these details are explained before moving on to explain the struct used during runtime.

1 ⟨Customization and global variables 1⟩≡

```
(defgroup whyse nil
  "noWeb HYpertext System in Emacs"
  :tag "WHYSE"
  :group 'applications)

(defcustom w-registered-projects nil
  "This variable stores all of the projects that are known to WHYSE."
  :group 'whyse
  :type '(repeat w--project-widget)
  :require 'widget
  :tag "WHYSE Registered Projects")
```

Defines:
`w-registered-projects`, used in chunks 1 and 3.
`whyse`, used in chunks 1, 4a, 1, 10c, 1, 23, 1, and 25.

The `w--project-widget` type used for the registered projects variable is a simple list widget containing the name of the project and a path to its Noweb source file, along with a shell command which generates the Noweb tool syntax for this project. Each Noweb project has a different command-line, and some are complex enough to have a makefile, or multiple makefiles! Noweb itself is an example of that level of complexity. The shell command is executed by WHYSE upon loading the project, and the standard output captured for parsing by a PEG parser.

The widget itself is explained in 1.1.

1.1 Widgetry

2 ⟨Widgets 2⟩≡

```
(define-widget 'w--project-widget 'list
  "The WHYSE project widget type."
  :format "\n%v\n"
  :offset 0
  :indent 0

  ;; NOTE: the convert-widget keyword with the argument
  ;; 'widget-types-convert-widget is absolutely necessary for ARGS to be
  ;; converted to widgets.
  :convert-widget 'widget-types-convert-widget
  :args '((editable-field
           :format "%t: %v"
           :tag "Name"
           :value "")

          (file
           :tag "Noweb source file (*.nw)"
           :format "%t: %v"
           :valid-regexp ".*\\.nw$"
           :value "")

          (string
           :tag "A shell command to run a shell script to generates Noweb tool syntax"
           :format "%t: %v"
           :documentation "A shell script which will produce the
           Noweb tool syntax. Any shell commands involved with
           noweave should be included, but totex should of course
           be excluded from this script. The script should output
           the full syntax to standard output. See the Noweb
           implementation of WHYSE for explanation."
           :value "")

          (editable-list
           :tag "LaTeX sectioning commands"
           :format "%t\n%v%i\n"
           :documentation "The strings here should be the maximum length of the command
                  possible, because substrings like 'section' will not match the
                  sectioning command only. It is likely that these will have false
                  positive matches (Sectioning command might be called like
                  '\\section{}' or '\\mySectionCommand[bold]{name}'.)"
           :args ((string :tag "Sectioning command"
                          :format "%t: %v"
                          :value "\\section{"
                          ;; FIXME: even with regex-builder I can't seem
                          ;; to write a regular expression that doesn't
                          ;; reject seemingly valid strings.
                          ;; :valid-regexp "\\.*\\(\\[.*\\]\\)?{?"
                          :documentation "More than simply 'section', use '\\section{' or '\mySection[]{' at least."
                          )))
          ))

(define-widget 'w--module-header 'group
```

```
    "Code and docs module editor headerbar.

  This widget represents the headerbar for the code and docs module
  editor."
    ⟨module-header top-level 28b⟩)
```

The definition of ⟨module-header top-level 28b⟩ is saved for later, because the organization of the editing buffer is experimental and therefore kept in the developer-oriented miscellany section C.3.

An example of what the list generated from the information entered into Customize would look like is given here for elucidation (as it would exist in a `custom-set-variables` form).

```
'(w-registered-projects
  '(("noWeb HYpertext System in Emacs"
     "~/Desktop/whyse.nw"
     "make -C ~/Desktop --silent --file ~/src/whyse/Makefile tool-syntax"))
  nil
  (widget))
```

The function documentation string should be expalnatory enough for the behaviour of the `whyse` command.

3 ⟨WHYSE 3⟩≡

```
(defun whyse ()
  "Opens the default whyse project, conditionally running hooks.

Hooks are only run if a project is actually opened. If
`w-load-default-project?' and
`w-open-customize-when-no-project-defined?' are both nil then a
warning is given and hooks are not run.

When both customization variables are non-nil, or if only
`w-load-default-project?' is nil, then Customize is opened to the
whyse group."
  (interactive)
  ;; Warn the user that their customization options have made `whyse' a
  ;; no-op function.
  (when (and (not w-load-default-project?)
             (not w-open-customize-when-no-projects-defined?))
    (warn "The customization options for `whyse' have effectively disabled the `whyse' command."))
  (if-let ((w-load-default-project?)
           (default-project (cl-first w-registered-projects))
           (project (make-w-project :name (cl-first default-project)
                                    :noweb (cl-second default-project)
                                    :script (cl-third default-project)))
           (parse-tree ⟨parse-project-in-temp-buffer 14c⟩))
      (progn
        ⟨setup project database 5b⟩
        (run-hooks 'w-open-project-hook)
        ;; TODO: connection closure should be on a timer that is reset
        ;; everytime a `w-'prefixed function is called, or something.
        (emacsql-close (w-project-database-connection project)))
    (unless (not w-open-customize-when-no-projects-defined?)
      (customize-group 'whyse))))
```

Defines:
`whyse`, used in chunks 1, 4a, 1, 10c, 1, 23, 1, and 25.
Uses `w-load-default-project?` 4a, `w-open-customize-when-no-projects-defined?` 4a, and `w-registered-projects` 1.

4a ⟨Customization and global variables 1⟩+≡

```
(defcustom w-load-default-project? t
  "Non-nil values mean the system will load the default project.

nil will cause the interactive command `whyse' to open Customize on
its group of variables."
  :type 'boolean
  :group 'whyse
  :tag "Load default project when `whyse' is invoked?")

(defcustom w-open-customize-when-no-projects-defined? t
  "Non-nil values mean the system will open Customize as necessary.

nil will cause `whyse' to simply do nothing when no project is
defined."
  :type 'boolean
  :group 'whyse
  :tag "Open Customize to the whyse group when `whyse' is invoked and no projects are defined?")
```

Defines:
`w-load-default-project?`, used in chunk 3.
`w-open-customize-when-no-projects-defined?`, used in chunk 3.
Uses `whyse` 1 3.

The structure accessed in the namesake command of the package is rather simple. TODO Ensure that the previous statement in-prose [not in the TODO summary] is still correct. It is defined quickly, then explained briefly.

4b ⟨WHYSE project structure 4b⟩≡

```
(cl-defstruct w-project
  "A WHYSE project"
  ;; Fundamental
  name
  noweb
  script
  sectioning-commands-regexs
  database-file
  database-connection

  ;; Usage
  frame

  ;; Metadata
  (date-created (current-time-string))
  date-last-edited
  date-last-exported

  ;; TODO: limit with a customization variable so that it does not grow too large.
  history-sql-commands)
```

Instances of this struct are only initialized with a few values: `name`, `noweb`, and `script`. The rest of the fields either have default values dependent upon the input data (like the database-file, database-connection, and date-created), or are given values when appropriate later in operation (such as `date-last-exported`) or upon initialization (`frame`).

Initialization when the interactive command is called is covered next; to summarize: `w-open-project-hook` is run.

TODO Describe initialization of the system after parsing.

1.2 Database initialization

TODO finish the creation of a database. Use what I learned in the fall!

Every project should have a database file located somewhere within the user's Emacs directory; if the user is a Spacemacs user, then Spacemacs' cache directory is used, otherwise the database is made in the user's Emacs directory and not a sub-directory thereof.

The form used to create the absolute path for the location of the database joins three things: the user's Emacs directory, `nil` or Spacemacs' cache directory, and the name of the project with ".db" appended. Note that concatenating `nil` with a string is the same as returning the string unchanged.

5a ⟨return a filename for the project database 5a⟩≡

```
(file-name-concat
 ;; Usually ~/.emacs.d/
 user-emacs-directory
 ;; `nil' or the Spacemacs cache directory.
 (when (file-directory-p (expand-file-name ".cache" user-emacs-directory))
   ".cache")
 ;; PROJECT-NAME.db
 (concat (w-project-name project)
         ".db"))
```

For SQLite, the path name of the database to connect to or create is sufficient to establish a connection, so the next step is to connect to the database and store the connection object in the appropriate slot of the project struct.

5b ⟨setup project database 5b⟩≡

```
⟨create a database connection 5c⟩
⟨map over SQL s-expressions, creating the tables 6a⟩
```

5c ⟨create a database connection 5c⟩≡

```
(setf (w-project-database-connection project)
      (emacsql-sqlite ⟨return a filename for the project database 5a⟩
                      ;; TODO: make this debug log buffer into a package
                      ;; customization.
                      :debug (with-current-buffer
                                 (get-buffer-create "emacsql-sqlite-debug-log")
                               (erase-buffer)
                               (current-buffer))))
```

The only thing left to do is establish the schema of the tables, which is done by mapping over several EmacSQL s-expressions.

6a ⟨map over SQL s-expressions, creating the tables 6a⟩≡

```
(mapcar (lambda (expression)
          (emacsql (w-project-database-connection project) expression))

        ;; A list of SQL s-expressions to create the tables.
        '([:create-table :if-not-exists module
           ([modulename content filename sectionname displacement
             modulenumber]
            (:primary-key [modulenumber modulename filename]))]

          [:create-table :if-not-exists parentchild
           ([parent child]
            (:primary-key [parent child]))]

          ;; NOTE: Identifier_used_in_module is a relation with the
          ;; attributes identifier_name and module_number. An identifier
          ;; can be used several different ways in each module that it
          ;; is in. It can be defined, referenced or modified. Each of
          ;; this functions correspond to different value of the
          ;; attribute type_of_usage. The type_of_usage attribute can
          ;; have three values: modified, referenced, and defined.
          ;; Line_number is an attribute of each relationship. All
          ;; attributes are included in the key.
          [:create-table :if-not-exists identifierusedinmodule
           ([identifiername modulenumber linenumber typeofusage]
            (:primary-key [identifiername modulenumber typeofusage]))]

          ;;; HOLD: the concept of "topic" is not present in Noweb, but
          ;;; is present in WEB. Consult CWEB documentation and the
          ;;; original WEB documenation to see the meaning of this
          ;;; table.
          ;; NOTE: Toplc_referenced_in_module is a relation with
          ;; attributes topic_name and module_number. The same topic may
          ;; be in several modules and each module can contain several
          ;; topics. Both attributes compose the primary key.
          [:create-table :if-not-exists topicreferencedinmodule
           ([topicname modulenumber]
            (:primary-key [topicname modulenumber]))]))
```

1.3 Customizing the behaviour of `whyse` with hooks

WHYSE is meant to be customizable, defining as little as necessary to implement a development environment for Noweb as described by (CITE Brown and Czejdo).

6b ⟨open-project-hook 6b⟩≡

```
(defvar w-open-project-hook '()
  "Hooks to run when `whyse' has opened a project.")
```

Defines:
`w-open-project-hook`, used in chunk 6b.

The default behaviour of WHYSE is to insert all the chunks of the parsed document into a database. Before it does that it works upon the parse tree, preparing it into a suitable format usable with EmacSQL (which the author is aware he's stated elsewhere).

6c ⟨default hook functions 6c⟩≡

```
(add-hook 'w-open-project-hook 'w--prepare-sexp-sql-from-file-tokens)
```

Chapter 2

Parsing project nowebs

This section covers the parsing of the noweb tool syntax produced when `whyse` executes the project's defined shell script to generate the tool syntax.

The peg package provides automatic parser generation from a formal PEG grammar. The grammar is based off of the description of the tool syntax given in the CITE Noweb Hacker's Guide.

2.1 PEG rules

Every character of an input text to be parsed by parsing expressions in a PEG must be defined in terminal rules of the formal grammar. The root rule in the grammar for Noweb tool syntax is the appropriately named `noweb` rule. Beginning `with-peg-rules` brought into scope, the root rule `noweb` is ran on the buffer containing the tool syntax produced by the project shell script.

The grammar can be broken into five sections, each covering some part of parsing.

7a ⟨PEG rules 7a⟩≡
⟨high-level Noweb tool syntax structure 7b⟩
⟨files and their paths 8a⟩
⟨chunks and their boundaries 8b⟩
⟨quotations 10d⟩
⟨keyword definitions 10e⟩
⟨meta rules 7c⟩

As stated, the `noweb` rule defines the root expression—or starting expression—for the grammar. The tool syntax of Noweb is simply a list of one or more files, which are each composed of at least one chunk. Ergo, the following ⟨high-level Noweb tool syntax structure 7b⟩ is defined.

7b ⟨high-level Noweb tool syntax structure 7b⟩≡

```
;;; Overall Noweb structure
(noweb (bob) (not header) (+ file) (not trailer) (eob))
```

It is a fatal error for WHYSE if the header or trailer wrapper keywords appear in the text it is to parse. They are totally irrelevant, and only matter for the final back-ends (TEX, LATEX, or HTML) that produce human-readable documenation.

The grammar needs to address the fact that the syntax of the Noweb tool format is highly line-oriented, given the influence of AWK on the design and usage of Noweb (a historical version was entirely implemented in AWK). The following ⟨meta rules 7c⟩ define rules which organize the constructs of a line-oriented, or data-oriented, syntax.

7c ⟨meta rules 7c⟩≡

```
;; Helpers
(nl (eol) "\n")
(!eol (+ (not "\n") (any)))
(spc " ")
```

TODO: Review the following paragraph and rephrase it.

With the ⟨meta rules 7c⟩ enabling easier definitions of what a given "keyword" looks like, the concept of a file needs to be defined. A file is "anything that looks like a file to Noweb". However, by default, only the chunk named "*" (it's chunk header is `<<*>>`) is tangled when no specific root chunk is given on the command line.

TODO: Write about the need for the overall document to be separate from the one-or-more files specified in the document. Exempli gratia: the current document, contained in `whyse.nw` contains two files, though they are separately tangled: `whyse.el` and `test-parser-with-temporary-buffer.el`. If these two files were tangled at the same time, such that the output file discovery ability of Noweb was used, then the there would be more than one file in the intermediate tool syntax, but still a single preceeding documentation chunk before the first file, and a single succeeding documentation chunk after the last file.

8a ⟨files and their paths 8a⟩≡

```
;; Technically, file is a tagging keyword, but that classification only
;; makes sense in the Hacker's guide, not in the syntax.
(file (bol) "@file" spc (substring path) nl
      (action (setq w--file-current-line 0))
      (list (and (+ chunk)
                 (list (or (and x-chunks i-identifiers)
                           (and i-identifiers x-chunks))))
            ;; Trailing documentation chunk and new-lines after the xref
            ;; and index.
            (opt chunk)
            (opt (+ nl)))
      `(path chunk-list -- (cons path chunk-list)))
(path (opt (or ".." ".")) (* path-component) file-name)
(path-component (and path-separator (+ [word])))
(path-separator ["\\/"])
(file-name (+ (or [word] ".")))
```

NOTE: Writing PEXes for matching file names was the most difficult part I have encountered so far, as it has forced me to understand that a first reading of documentation is usually not sufficient to understand a complex library in an area of programming I have not practiced in before (language parsing).

Because chunks must not overlap, but can nest, the beginnings of chunks need to be pushed to the parsing stack and the end of a chunk needs to be popped off of it. The stack pushing operations in `kind` and `ordinal` delimit chunks by their kinds and number, and the stack actions in the `end` rule check that the chunk-related tokens on the stack are balanced.

8b ⟨chunks and their boundaries 8b⟩≡

```
(chunk begin (list (* chunk-contents)) end)
(begin (bol) "@begin" spc kind spc ordinal (eol) nl
       (action (if (string= (cl-second peg--stack) "code")
                   (setq w--peg-parser-within-codep t))))
(end (bol) "@end" spc kind spc ordinal (eol) nl
     (action
      (setq w--peg-parser-within-codep nil))
     ;; The stack grows down and the heap grows up,
     ;; that's the yin and yang of the computer thang
     `(kind-one
       ordinal-one
       keywords
       kind-two
       ordinal-two
       --
       (if (and (= ordinal-one ordinal-two) (string= kind-one kind-two))
           (cons (cons (if (string= kind-one "code")
                           'code
                         'docs)
                       ordinal-one)
                 keywords)
         (error "Chunk nesting error encountered."))))
(ordinal (substring [0-9] (* [0-9]))
         `(number -- (string-to-number number)))
(kind (substring (or "code" "docs")))
```

Valid `chunk-contents` is somewhat confusing, because chunks can contain many types of information other than text and new lines. The definition of what is valid follows.

1. `text`
2. `nl`
3. `defn` *name*
4. `use` *name*
5. `line` *n*
6. `language` *language*
7. `index …`
8. `xref …`

Any other keywords are invalid inside a code block. An example of an invalid keyword is anything related to quotations! This restriction only applies to code blocks, however, and documentation chunks may contain quotations, of course. As an exception, the keywords were originally banned inside code chunks, but to parse the noweb document in which WHYSE itself was written it needed to be adjusted. The grammar should be studied again to ensure that textual description and reality are in step.

9a ⟨chunks and their boundaries 8b⟩+≡
```
(chunk-contents
 (or
  ⟨structural keywords 9c⟩
  ⟨tagging keywords 9d⟩
  x-notused
  ⟨tool errors 10a⟩))
```

It is easier to handle the fatal keyword appearing inside chunks when it is a permissible keyword to appear inside a chunk; this allows the parser to consider a chunk with fatal inside of it as a valid chunk, but that does not mean that a chunk with a fatal keyword inside it does not invalidate a Noweb, it still does: the fatal keyword causes a fatal crash in parsing regardless. Those structural keywords which may be used inside the contents of a chunk are given next.

9b ⟨structural keywords (except quotations) 9b⟩≡
```
;; structural
text
nwnl ;; Noweb's @nl keyword, as differentiated from the rule nl := "\n".
defn
;;; NOTE: previously, a note on the following line incorrectly stated
;;; the `use' token was related to the `identifierusedinmodule' table,
;;; when it is actually related to the `parentchild' table.
use
```

All structural keywords, then, are:

9c ⟨structural keywords 9c⟩≡
```
⟨structural keywords (except quotations) 9b⟩
quotation
```

9d ⟨tagging keywords 9d⟩≡
```
;; tagging
line
language
;; index
i-define-or-use
i-definitions
;; xref
x-prev-or-next-def
x-continued-definitions-of-the-current-chunk
i-usages
x-usages
x-label
x-ref
```

TODO Verify that this statement is true: "Usually Noweb will warn a user that a chunk was referenced but undefined, or that there was some other issue with chunks." Sometimes, however, the system will permit a chunk to be undefined and this leads to the only cases in the tool syntax where it is not line-oriented. `noidx` will read the cross references to other chunks and will be unable to generate the label, so it will insert `@notdef` where it would otherwise upcase "nw" and then insert the label. This is why `x-undefined` is placed among the other ⟨tool errors 10a⟩ keywords.

10a ⟨tool errors 10a⟩≡
```
;; error
fatal
x-undefined
```

The fundamental keywords are text and nwnl (new line, per Noweb convention). Text keywords contain source text, and any new line tokens in the source text are replaced with the appropriate number of @nl keywords (per convention); these are reduced to a single text token when they are adjacent on the `peg--stack`.

10b ⟨chunks and their boundaries 8b⟩+≡
```
(text (bol) "@text" spc (substring (* (and (not "\n") (any)))) nl
      `(txt -- (w--concatenate-text-tokens (cons 'text txt))))
(nwnl (bol) (substring "@nl") nl (action (setf w--file-current-line
                                               (1+ w--file-current-line)))
      ;; Be sure that when thinking about the symbol `nl' here that
      ;; you're not confusing it with the peg rule nl.
      `(nl -- (w--concatenate-text-tokens (cons 'nl "\n"))))
```
Uses `w--concatenate-text-tokens` 18b.

Nowebs are built from chunks, so the definition and usage of (i.e. references to) a chunk are important keywords.

10c ⟨chunks and their boundaries 8b⟩+≡
```
(defn "@defn" spc (substring !eol) nl
  `(name -- (cons 'chunk name)))

;; In <<whyse.el>>=, it leads to usages tokens like below:
;; (chunk-child-usage . "Commentary")
;; (chunk-child-usage . "Code")
(use
 (bol) "@use" spc (substring !eol) nl
 `(name --
        (if name (cons 'chunk-child-usage name)
          (error "UH-OH! There's a syntax error in the tool output!"))))
```
Uses `whyse` 1 3.

Documentation may contain text and newlines, represented by @text and [@nwnl]. It may also contain quoted code bracketed by @quote . . . @endquote. Every @quote must be terminated by an @endquote within the same chunk. Quoted code corresponds to the … construct in the noweb source.

10d ⟨quotations 10d⟩≡
```
(quotation (bol) "@quote" nl
           (action (when w--peg-parser-within-codep
                     (error "The parser found a quotation within a code chunk. A @fatal should have been found here, bu
           (substring (+ (and (not "@endquote") (any))))
           (bol) "@endquote" nl
           `(lst -- (cons 'quotation lst)))
```

10e ⟨keyword definitions 10e⟩≡
```
(line (bol) "@line" spc (substring ordinal) nl
      `(o -- (cons 'line o)))

(language (bol) "@language" spc (substring words-eol))
```

The indexing and cross-referencing abilities of Noweb are excellent features which enable a reader to navigate through a printed (off-line) or on-line version of the literate document quite nicely. These functionalities each begin with a rule which matches only part of a line of the tool syntax since there are many indexing and cross-referencing keywords. The common part of each line is a rule which merely matches the `@index` or `@xref` keyword. The rest of the lines are handled by a list of rules in `index-keyword` or `xref-keyword`.

The Noweb Hacker's Guide lists these two lines in the "Tagging keywords" table, indicating that it's unlikely (or forbidden) that the index or xref keywords would appear alone without any subsequent information on the same line.

> @index ... Index information.
> @xref ... Cross-reference information

There are many keywords defined by the Noweb tool syntax, so they are referenced in this block and defined and documented separately. Some of these keywords are delimiters, so they are not given full "keyword" status (defined as a PEX rule) but exist as constants in the definition of a rule that defines the grouping.

11a ⟨keyword definitions 10e⟩+≡

```
;; Index
⟨indexing and cross-referencing set-off words 11b⟩
⟨fundamental indexing keywords, which are restricted to within a code chunk 11c⟩
⟨the index of identifiers 12c⟩
⟨unsupported indexing keywords 12d⟩

;; Cross-reference
⟨cross-referencing keywords 13⟩

;; Error
⟨error-causing keywords 14a⟩
```

Further keywords are categorized neatly as Indexing or Cross–referencing keywords, so they are contained in subsections.

2.1.1 indexing

Indexing keywords, both those used within chunks and those used outside of chunks, are defined in this section. The `<<fundamental indexing keywords, which are restricted to within a code chunk>>`, index definitions or usages of identifiers and track the definitions of identifiers in a chunk and the usages of identifiers in a chunk. They may seem redundant, but are not; the Noweb Hacker's Guide offers a better explanation of the differences.

11b ⟨indexing and cross-referencing set-off words 11b⟩≡

```
(idx (bol) "@index" spc)
(xr (bol) "@xref" spc)
```

11c ⟨fundamental indexing keywords, which are restricted to within a code chunk 11c⟩≡

```
(i-define-or-use
 idx
 (substring (or "defn" "use")) spc (substring !eol) nl
 (action
  (unless w--peg-parser-within-codep
      (error "WHYSE parse error: index definition or index usage occurred outside of a code chunk.")))
 `(s1 s2 -- (cons (make-symbol s1) s2)))

⟨identifiers defined in a chunk 12a⟩
⟨identifiers used in a chunk 12b⟩
```

12a ⟨identifiers defined in a chunk 12a⟩≡
```
(i-definitions idx "begindefs" nl
               (list (+ (and (+ i-isused) i-defitem)))
               idx "enddefs" nl
               `(definitions -- (cons 'definitions definitions)))
(i-isused idx (substring "isused") spc (substring label) nl
          `(u l -- (cons 'used! l)))
(i-defitem idx (substring "defitem") spc (substring !eol) nl
           `(d i -- (cons 'def-item i)))
```

12b ⟨identifiers used in a chunk 12b⟩≡
```
(i-usages idx "beginuses" nl
          (list (+ (and (+ i-isdefined) i-useitem)))
          idx "enduses" nl
          `(usages -- (cons 'usages usages)))
(i-isdefined idx (substring "isdefined" spc label) nl)
(i-useitem idx (substring "useitem" spc !eol) nl) ;; !eol :== ident
```

The summary index of identifiers is a file–specific set of keywords. The index lists all identifiers defined in the file (at least all of those recognized by the autodefinitions filter).

12c ⟨the index of identifiers 12c⟩≡
```
(i-identifiers idx "beginindex" nl
               (list (+ i-entry))
               idx "endindex" nl
               `(l -- (cons 'i-identifiers l)))
(i-entry idx "entrybegin" spc (substring label spc !eol) nl
         (list (+ (or i-entrydefn i-entryuse)))
         idx "entryend" nl
         `(entry-label lst -- (cons 'entry-label lst)))
(i-entrydefn idx (substring "entrydefn") spc (substring label) nl
             `(defn label -- (cons 'defn label)))
(i-entryuse idx (substring "entryuse") spc (substring label) nl
            `(use lst -- (cons 'use lst)))
```

The following chunk's name is documentation enough for the purposes of WHYSE. See the Noweb Hacker's Guide for more information.

`@index nl` was deprecated in Noweb 2.10, and `@index localdefn` is not widely used (assumedly) nor well-documented, so it is unsupported by WHYSE (contributions for improved support are welcomed).

12d ⟨unsupported indexing keywords 12d⟩≡
```
;; @index nl was deprecated in Noweb 2.10, and @index localdefn is not
;; widely used (assumedly) nor well-documented, so it is unsupported by
;; WHYSE (contributions for improved support are welcomed).
(i-localdefn idx "localdefn" spc !eol nl)
(i-nl idx "nl" spc !eol nl
      (action (error ⟨index nl error message 14b⟩)))
```

2.1.2 cross referencing

13 ⟨cross-referencing keywords 13⟩≡

```
(x-label xr (substring "label" spc label) nl
         `(substr -- (cons 'x-label (cadr (split-string substr)))))
(x-ref xr (substring "ref" spc label) nl
       `(substr --  (cons 'ref (cadr (split-string substr)))))

;; FIXME: improve the error handling at this point. It is not fragile
;; any longer, becasue most things are ignored and this is hackish;
;; however, the message reporting is not too helpful. It would be nice
;; to have _only_ the chunk name reported, and formatted with << and >>.
;;; Reproduction steps: make a reference to an undefined code chunk
;;; within another code chunk. For fixing this issue, undefined code
;;; chunks should also be referenced within quotations in documentation.
(x-undefined
 xr (or "ref" "chunkbegin") spc
 (guard
  (if (string= "nw@notdef"
               (buffer-substring-no-properties (point) (+ 9 (point))))
      (error (format "%s: %s: %s:\n@<@<%s>>"
                     "WHYSE"
                     "nw@notdef detected"
                     "an undefined chunk was referenced"
                     (buffer-substring-no-properties (progn (forward-line) (point))
                                                     (end-of-line)))))))

(x-prev-or-next-def
 xr (substring (or "nextdef" "prevdef")) spc (substring label) nl
 `(previous-or-next-chunk-defn label -- (cons (make-symbol previous-or-next-chunk-defn) label)))

(x-continued-definitions-of-the-current-chunk
 xr "begindefs" nl
 (list (+ (and xr (substring "defitem") spc (substring label) nl)))
 xr "enddefs" nl)

(x-usages
 xr "beginuses" nl
 (list (+ (and xr "useitem" spc (substring label) nl)))
 xr "enduses" nl)

(x-notused xr "notused" spc (substring !eol) nl
           `(name -- (cons 'unused! name)))
(x-chunks nwnl
          nwnl
          xr "beginchunks" nl
          (list (+ x-chunk))
          xr "endchunks" nl
          `(l -- (cons 'x-chunks l)))
(x-chunk xr "chunkbegin" spc (substring label) spc (substring !eol) nl
         (list (+ (list (and xr
                             (substring (or "chunkuse" "chunkdefn"))
                             `(chunk-usage-or-definition -- (make-symbol chunk-usage-or-definition))
                             spc
                             (substring label)
                             nl))))
         xr "chunkend" nl)

;; Associates label with tag (@xref tag $LABEL $TAG)
(x-tag xr "tag" spc label spc !eol nl)
(label (+ (or "-" [alnum]))) ;; A label never contains whitespace.
```

14a ⟨error-causing keywords 14a⟩≡
```
;; User-errors (header and trailer) and tool-error (fatal)
;; Header and trailer's further text is irrelevant for parsing, because they cause errors.
(header (bol) "@header" ;; formatter options
        (action (error "[ERROR] Do not use totex or tohtml in your noweave pipeline.")))
(trailer (bol) "@trailer" ;; formatter
         (action (error "[ERROR] Do not use totex or tohtml in your noweave pipeline.")))
(fatal (bol) "@fatal"
       (action (error "[FATAL] There was a fatal error in the pipeline. Stash the work area and submit a bug report aga
```

14b ⟨index nl error message 14b⟩≡
```
(string-join
 '("\"@index nl\" detected."
  "This indicates hand-written @ %def syntax in the Noweb source."
  "This syntax was deprecated in Noweb 2.10, and is entirely unsupported."
  "Write an autodefs AWK script for the language you are using.")
 "\n")
```

14c ⟨parse-project-in-temp-buffer 14c⟩≡
```
(with-temp-buffer
  (insert (shell-command-to-string (w-project-script project)))
  (goto-char (point-min))
  (w--parse-current-buffer-with-rules))
```

Uses `w--parse-current-buffer-with-rules` 15a.

14d ⟨with-project 14d⟩≡
```
;; TODO: understand how and why this macro works like
;; `with-temp-buffer'; I only copied a couple things from the defintion
;; of that macro, but I don't yet understand its definition fully. I
;; will need to before I really start to understand Emacs Lisp. I do
;; understand enough, but there is a lot more to know. A whole lot.
(defmacro w-with-project (project &rest body)
  (declare (indent 0) (indent 2) (debug t))
  "Evaluates BODY with PROJECT in scope. This is like `let' except
it doesn't bind arbitrary values to arbitrary symbols. PROJECT is
taken as the symbol `project' during evaluation of BODY.

PROJECT is in scope because it is an argument of this macro. This
may or may not work as expected."
  (eval `(progn ,@body)))
```

Defines:
`w-with-project`, never used.

15a ⟨buffer parsing function 15a⟩≡

```
;; FIXME: the current parse tree contains a `nil' after the chunk type
;; and number assoc, and that needs to be analyzed. Why is this `nil' in
;; the stack? I assume and believe it is because of the collapsing of
;; stringy tokens; when a token should be put back onto the stack it may
;; also be putting a `nil' onto the stack in the first call to the
;; function.
;;;; Parsing expression grammar (PEG) rules
(defun w--parse-current-buffer-with-rules ()
  "Parse the current buffer with the PEG defined for Noweb tool syntax."
  (with-peg-rules
      (⟨PEG rules 7a⟩)
    (let (w--peg-parser-within-codep
          (w--first-stringy-token? t))
      (peg-run (peg noweb) #'w--parse-failure-function))))

(defun w--parse-failure-function (lst)
  (setq w--parse-success nil)
  (pop-to-buffer (clone-buffer))
  (save-excursion
    (put-text-property (point) (point-min)
                       'face 'success)

    (put-text-property (point) (point-max)
                       'face 'error)

    (goto-char (point-max))
    (message "PEXes which failed:\n%S" lst)))
```

Defines:
  `w--parse-current-buffer-with-rules`, used in chunks 14c and 25.
  `w--parse-failure-function`, never used.
Uses `w--parse-success` 15b 25.

15b ⟨Customization and global variables 1⟩+≡

```
(defvar w--parse-success t
  "The success or failure of the last parsing of noweb tool syntax.")
```

Defines:
  `w--parse-success`, used in chunk 15a.

Chapter 3

Processing parsed nowebs into SQL

This section covers how the parsed text generated in the last section is processed, creating a series of SQL statements that will be executed by SQLite using the interface provided by the EmacSQL package.

First, the overall structure of the parsed text should be diagrammed. The parse tree is a list of noweb documents, each being a list themselves. The first atom of an inner list, corresonding to a document, is the filename of that document (hopefully the same filename as passed on the commmand-line elsewhere when the document is used).

Deeper, each document-list contains as the second atom a list of its contents, which is an association list thereof. Each association in the alist is bewteen the symbol relating to the rule that generated the cons cell, and the contents appropriate to the symbol.

3.1 Processing

There are many steps to compiling the parse tree into SQL. The first step is to ensure the association lists in the parse tree are in a format that is acceptable to built-in Emacs Lisp functions; this will make it easier to navigate the tree and transform it. Other texts call this manipulation of the parse tree "list destructuring"; it could be thought of as code generation, since the data is used to generate DML which is partly code and partly data.

Some associations are reductions from the initial parse tree—which thanks to the peg package and PEG parsing in general—are not reduced in a second step. Modifications to the parsing procedure can occur directly without losing information, and thanks to literate programming should be easy for advanced users.

The initial parse tree begins like the one below.

```
'((noweb-document-one
   ((docs . 0)
    (text . "\tex{} is cool!"))
   ((code . 1)
    (text . "(message \"LISP is awesome!\")")))
  (noweb-document-two
   ((code . 0)
    (text . "asdf is a system definition format in Common LISP,"))
    (nwnl . "\n")
    (text . "and I like to use it.")
   ((code . 1)
    (text . "jkl; is the right-handed corollary of asdf."))
   ((docs . 2)
    (text . "\latex{} is great!"))
   ((docs . 3)
    (text . "Noweb, written by Norman Ramsey is sweet!"))))
```

The reductions which occur during parsing make chunk zero of the second noweb document look like this in the final result:

```
((code . 0)
 (text .
  "asdf is a system definition format in Common LISP,
  and I like to use it."))
```

; the new result is much easier to use as data for other programs (SQL in this case). In the verbatim text a literal newline was inserted rather than retaining the escape sequence, which is exactly what happens in the reduction step as well. The next subsection discusses the details of how the reduction in complexity exampled above is achieved.

3.1.1 Reducing complexity in the alist

The first step in making the parse tree navigable for other programs is collapsing adjacent "stringy" tokens into single `text` tokens. The output tool syntax of notangle, and the parse tree resulting from the PEG, (breifly) contain individual text tokens for fragments of whole text lines and form feed characters. These tokens exist because the cross-referencing tokens fragment the text lines, and new lines in the noweb document are treated specially to facilitate this fragmentation.

A small quote from the tool syntax of a development version of WHYSE is shown in this example in its parsed form. However, during actual parsing these adjacent tokens are immediately collapsed into singular tokens.

```
(text . "  and \textsc{Noweb}'s \texttt{finduses.nw}!")
(nwnl . "@nl")
(text . "\end{enumerate}")
(nwnl . "@nl")
(text . "")
(nwnl . "@nl")
```

To collapse these tokens into a single text token the ***peg--stack*** must be manipulated carefully. It isn't advisable to manipulate this variable in the course of a PEG grammar's actions. There is a use case for it when the previous rules and actions won't accomodate the necessary action without refactoring a larger part of the grammar. In this development version that is not a goal; basic functionality is sought after, not robustness or beauty, so hacking the desired behaviour together quickly is better.

`w--nth-chunk-of-nth-noweb-document` retrieves the parse tree for the nth noweb document, which in the case of `whyse.nw` is the parse tree of the zeroth-indexed document. It's quite a simple function. To obtain a given chunk of this document from the parse tree the result of the function is called with `nth` and the index of the chunk.

17 ⟨functions for navigating WHYSE parse trees 17⟩≡

```
(defun w--nth-document-file-name (nth-document parse-tree)
  "Return the file name of the nth-indexed document in the parse tree.

For the first document in the parse tree, that is the
zeroth-indexed document."
  (cl-first (nth nth-document parse-tree)))

(defun w--nth-document (nth-document parse-tree)
  "Return the subtree of the nth-indexed document in the parse tree."
  (cl-second (nth nth-document parse-tree)))

(defun w--nth-chunk-of-document (n document)
  "Return the subtree for the Nth chunk of a noweb document parse subtree."
  (nth n document))

(defun w--chunk-number (chunk)
  "Return the chunk number of CHUNK."
  (or (cdr (assq 'code chunk))
      (cdr (assq 'docs chunk))))

(defun w--chunk-text (chunk)
  "Join all the strings returned from the collection in the loop,
and return the single string."
  (string-join
   (cl-loop for elt in chunk collect
            (when (and (listp elt) (equal 'text (car elt)))
              (cdr elt)))
   ""))

(defun w--chunk-name (chunk)
  "Return non-nil if CHUNK is a code chunk, and thereby has a name.

The return value, if non-nil, is actually the name of the chunk."
```

```
    (if-let ((name (assq 'chunk chunk)))
        (cdr name)))
```

Defines:
  w--chunk-name, used in chunk 19.
  w--chunk-number, used in chunk 19.
  w--chunk-text, used in chunk 19.
  w--nth-chunk-of-document, never used.
  w--nth-document, never used.
  w--nth-document-file-name, never used.

TODO: place this inclusion better

18a ⟨Code 18a⟩≡
```
⟨functions for navigating WHYSE parse trees 17⟩
⟨functions to collapse text and newline tokens into their largest possible form 18b⟩
```

18b ⟨functions to collapse text and newline tokens into their largest possible form 18b⟩≡
```
(defun w--concatenate-text-tokens (new-token)
  "Join the values of two text token associations in a two-element token alist.

If the two associations shouldn't be joined, return them to the stack."
  (prog1
      ;; Concatenation only occurs when the previous token examined was
      ;; a text or nwnl token, ergo there must have been a text or nwnl
      ;; token previously examined for any concatenation to occur. When
      ;; no such token has been examined immediately return the
      ;; (stringy) token recieved and indicate it must have been a
      ;; stringy token by chaning the value of `w--first-stringy-token?'
      ;; accordingly. Subsequent runs will then operate on potential
      ;; pairs of stringy tokens.
      (if-let ((not-first-stringy-token? (not w--first-stringy-token?))
               (previous-token  (pop peg--stack))
               ;; The previous token cannot be a text or nwnl token if
               ;; it is not a list, and checking prevents causing an
               ;; error by taking the `car' of a non-list token, e.g. the
               ;; filename token.
               (previous-token-is-alist?
                (prog1 (and (listp previous-token)
                            (listp new-token)
                            (or (assoc 'text `(,new-token))
                                (assoc 'nl `(,new-token)))
                            (or (assoc 'text `(,previous-token))
                                (assoc 'nl `(,previous-token)))))))
          ;; Join the association's values and let the caller push a single
          ;; token back onto the `peg--stack'.
          (cons 'text (format "%s%s" (cdr previous-token)
                              (cdr new-token)))

        ;; Push the previous token back to the `peg--stack', and let the
        ;; caller push the new token to that stack.
        (push previous-token peg--stack)
        new-token)
    (when w--first-stringy-token? (setq w--first-stringy-token? nil))))
```

Defines:
  w--concatenate-text-tokens, used in chunk 10b.

TODO Better document the functionality of the default hook function: `w--prepare-sexp-sql-from-file-tokens`. The current explanation is a copy of the brief explanation in the hook documentation string.

The 'parse-tree' from the lexical environment (or scope, given that WHYSE is not implemented with lexical binding) is mapped over until every file token has been processed and records have been inserted into all four tables of the database.

19 ⟨default hook functions 6c⟩+≡

```
(defun w--prepare-sexp-sql-from-file-tokens ()
  "Prepare an s-expression of SQL statements for `emacsql'.

The `parse-tree' from the lexical environment (or scope, given
that WHYSE is not implemented with lexical binding) is mapped
over until every file token has been processed and records have
been inserted into all four tables of the database.

This hook depends on this object being in scope: `parse-tree'.
That object is in scope when this hook runs with the default
implementation of `whyse'. (This is not meant to imply other
implementations exist [yet], only that hacked up installations
won't operate with any guarantees or according to the original
documentation. You know that, though, because you're reading the
source code and documentation for the original right now, and it
should be obvious that changing the source code should have been
accompanied with changes in the documentation.)"
  (mapcar
   (lambda (file-token)
     (let* ((file-name (car file-token))
            (chunks (cdr file-token))
            (modules (vector
                      :insert :into 'module
                      :values (mapcar #'w--vectorize-chunk-data chunks)
                      :on-conflict-do-nothing))
            (parent-child (vector
                           :insert :into 'parentchild
                           :values (w--parent-child-relationships)
                           :on-conflict-do-nothing))
            (connection (w-project-database-connection project)))
       (mapcar (lambda (seql) (emacsql connection seql))
               (list modules
                     parent-child
                     ;; identifier-used-in-module
                     ;; topic-referenced-in-module
                     ))))
   parse-tree))

(defun w--vectorize-chunk-data (chunk)
  "Convert an individual chunk to a vector of objects.

NOTE: order of arguments: [module-name module-text file-name
section displacement module-number]"
  (let ((chunk-name (w--chunk-name chunk)))
    (vector chunk-name
            (w--chunk-text chunk)
            file-name
            (when chunk-name
              (w--chunk-section chunk-name))
            nil
            (w--chunk-number chunk))))

(defun w--parent-child-relationships ()
  "Make a list of vectors of parent-child relationships.

The elements are taken from applying `w--get-chunk-uses-of-chunks' to
every chunk in the noweb."
  (apply #'append (cl-mapcar #'w--get-chunk-uses-of-chunks chunks)))
```

```
(defun w--get-chunk-uses-of-chunks (chunk)
  "Make a list of vectors of parent-child relationships.

The parent is CHUNK, and the first element of each vector in the
list will be the number of this chunk. The chunks this chunk uses
will be named in the vectors in the returned list."
  (with-temp-buffer
    (let* ((iter 0)
           (parent-uses-child (list nil))
           (chunk-number (assoc-default 'code chunk))
           (chunk-formatted (format "%S" chunk)))
      (insert chunk-formatted)
      (goto-char (point-min))
      (cl-remove
       nil
       (cl-mapcar (lambda (assoc)
                    (when (equal (car assoc) 'chunk-child-usage)
                      (vconcat (list (number-to-string chunk-number)
                                     (cdr assoc)))))
                  chunk)))))
```

Defines:
  `w--get-chunk-uses-of-chunks`, never used.
  `w--parent-child-relationships`, never used.
  `w--prepare-sexp-sql-from-file-tokens`, used in chunk 19.
  `w--vectorize-chunk-data`, never used.
Uses `w--chunk-name` 17, `w--chunk-number` 17, `w--chunk-section` 21, and `w--chunk-text` 17.

When the SEQL (S-EXP SQL) has been compiled, it should be logged. This might be better controlled with DCL (the data control langauge featureset of SQL). TODO Make the number of entries in this history list customizable, limited to some given number or a default of one hundred. The history of SEQL statements compiled is treated like a stack, with the newest statement being the first element of the list. A shoddy implementation of a "stack-based history" is given next.

20a ⟨push the compiled SQL to the database and to the history stack 20a⟩≡

```
;; NOTE: the result of evaluating the SQL is pushed to the history stack
;; alongside the SQL that was executed.
(cl-pushnew (cons (emacsql (w-project-database-connection default-project)
                           compiled-parse-tree)
                  . compiled-parse-tree)
            (w-project-history-sql-commands default-project))
```

3.1.2 finding the section title of a chunk

20b ⟨Code 18a⟩+≡

⟨find section title 21⟩

21 ⟨find section title 21⟩≡

```
;; TODO 2024-05-13: this function is really rough. It still needs a
;; working algorithm to search through the noweb text to find the
;; section that a chunk belongs to. I think that searching for all of
;; the sections that preceed a chunk definition will be necessary before
;; finding the last section command that preceeds a chunk (thereby
;; finding the section the chunk belongs to). There may be a better
;; algorithm; this is the first algorithm I thought of to find the
;; section that a chunk belongs to regardless of the regular expression
;; used.
;;
;; Another algorithm might use Emacs Lisp regular expressions to search
;; for the first match to any of the set of regular expressions that
;; would search for a given sectioning command.
;;
;; For now, I trust the Emacs Lisp manual when it says that `regex-opt'
;; is efficient; it is damned convenient right now.
(defun w--chunk-section (chunk-name)
  (with-temp-buffer
    (insert-file-contents (w-project-noweb project))
    (re-search-forward
     ;; It is an error if no chunks are found.
     (regexp-opt (list (concat "<" "<" chunk-name ">" ">" "="))))
    (re-search-backward
     (regexp-opt (w-project-sectioning-commands-regexs project))
     ;; It is not an error if no section is found because sectioning is
     ;; entirely optional in a literate document with LaTeX.
     nil t)
    (buffer-substring (line-beginning-position) (line-end-position))))
```

Defines:
w--chunk-section, used in chunk 19.

While it's certain that some of the data from a search operation are markers of a match (some of them are non-nil), substrings corresponding to the used chunk name in the current chunk are appended to a list that is ultimately returned.

3.1.3 Determining parent-child relationships

There needs to be a function which can take a generalizable approach to determining the name of a section defined by any possible sectioning command. To easily do that a function could be provided by the user for any sectioning commands that are exotic, and those functions will accommodate the LaTeX command while the calling function will handle finding them and collecting their output.

```
␣␣((code␣.␣32)
␣␣␣(x-label␣.␣"NW2ucZJx-3LzQog-2")
␣␣␣(ref␣.␣"NW2ucZJx-3LzQog-1")
␣␣␣(chunk␣.␣"chunks␣and␣their␣bou...")
␣␣␣(prevdef␣.␣"NW2ucZJx-3LzQog-1")
␣␣␣(nextdef␣.␣"NW2ucZJx-3LzQog-3")
␣␣␣(text␣.␣"\n(chunk-contents\n␣(o...")
␣␣␣(x-label␣.␣"NW2ucZJx-3LzQog-2-u1")
␣␣␣(ref␣.␣"NW2ucZJx-2ZphFz-1")
␣␣␣(chunk-child-usage␣.␣"structural␣keywords")
␣␣␣(text␣.␣"\n␣␣")
␣␣␣(x-label␣.␣"NW2ucZJx-3LzQog-2-u2")
␣␣␣(ref␣.␣"NW2ucZJx-44iV1g-1")
␣␣␣(chunk-child-usage␣.␣"tagging␣keywords")
␣␣␣(text␣.␣"\n␣␣x-notused\n␣␣")
␣␣␣(x-label␣.␣"NW2ucZJx-3LzQog-2-u3")
␣␣␣(ref␣.␣"NW2ucZJx-1ZiKLq-1")
␣␣␣(chunk-child-usage␣.␣"tool␣errors")
␣␣␣(text␣.␣"))\n"))
```

22 ⟨collect child chunk uses 22⟩≡

```
(with-temp-buffer
  (let* ((iter 0)
         (parent-uses-child (list nil))
         (chunk-number (assoc-default 'code chunk))
         (chunk-formatted (format "%S" chunk)))
    (insert chunk-formatted)
    (goto-char (point-min))
    (cl-remove
     nil
     (cl-mapcar (lambda (assoc)
                  (when (equal (car assoc) 'chunk-child-usage)
                    (vconcat (list chunk-number (cdr assoc)))))
                chunk))))
```

Appendix A

Packaging whyse

Installing an Emacs Lisp package is quite easy if the system is distributed through the GNU Emacs Lisp Package Archive (GNU ELPA), and only slightly less easy if it is distributed through MELPA (Milkypostman's Emacs Lisp Package Archive). Other package archives have existed, but they are all ephemeral. The most popular alternative to GNU ELPA, Non-GNU ELPA, and MELPA is direct distribution of files through Git servers and the use of a package by the end user to install directly from such.

This software is in-development, so it will only be distributed directly through Git.

WHYSE follows the form of "simple", single-file packages documented in the Emacs Lisp Reference Manual. The package file, *whyse.el*, is emitted by *notangle* which is called by the Makefile in every target but `clean`. All source development occurs in *whyse.nw* using Polymode.

The makefile distributed alongside whyse.nw in the tarball contains the command-line used to tangle and weave WHYSE.

23a ⟨whyse.el 23a⟩≡
```
⟨Emacs Lisp package headers 23b⟩
⟨Licensing and copyright 24a⟩
⟨Commentary 24b⟩
⟨Code 18a⟩
⟨provide the whyse feature and list the file local variables 24d⟩
```

23b ⟨Emacs Lisp package headers 23b⟩≡
```
;;; whyse.el --- noWeb HYpertext System in Emacs -*- lexical-binding: nil -*-
;; Yes, you read that right: no lexical binding in this file.

;; Copyright © 2023 Bryce Carson

;; Author: Bryce Carson <bcars268@mtroyal.ca>
;; Created 2023-06-18
;; Keywords: tools tex hypermedia
;; URL: https://github.com/bryce-carson/whyse

;; This file is not part of GNU Emacs.
```
Uses `whyse` 1 3.

23c ⟨whyse-pkg.el 23c⟩≡
```
(define-package "whyse" "0.1" "noWeb HYpertext System in Emacs"
  '(⟨required packages 23d⟩))
```
Uses `whyse` 1 3.

The following chunk lists the ⟨required packages 23d⟩; as of `whyse-0.1-devel` the only required packages are `peg` and `cl-lib`, `emacsql`.

23d ⟨required packages 23d⟩≡
```
(emacs "25.1")
(emacsql "20230220")
(peg "1.0.1")
(cl-lib "1.0")
```

The license text was included in this document using its LaTeX form in the License section at the end of this document.

24a ⟨Licensing and copyright 24a⟩≡

```
;; This program is free software: you can redistribute it and/or
;; modify it under the terms of the GNU General Public License as
;; published by the Free Software Foundation, either version 3 of the
;; License, or (at your option) any later version.

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

;; You should have received a copy of the GNU General Public License
;; along with this program. If not, see
;; <https://www.gnu.org/licenses/>.
```

24b ⟨Commentary 24b⟩≡

```
;;; Commentary:
;; WHYSE was described by Brown and Czedjo in _A Hypertext for Literate
;; Programming_ (1991).
;;
;; Brown, M., Czejdo, B. (1991). A hypertext for literate programming.
;;    In: Akl, S.G., Fiala, F., Koczkodaj, W.W. (eds) Advances in
;;    Computing and Information - ICCI '90. ICCI 1990. Lecture Notes in
;;    Computer Science, vol 468. Springer, Berlin, Heidelberg.
;;    https://doi-org.libproxy.mtroyal.ca/10.1007/3-540-53504-7_82.
;;
;; A paper describing this implementation---written in Noweb and browsable,
;; editable, and auditable with WHYSE, or readable in the printed form---is
;; hoped to be submitted to The Journal of Open Source Software (JOSS)
;; before the year 2024. N.B.: the paper will include historical
;; information about literate programming, and citations (especially
;; of those given credit here for ideating WHYSE itself).
```

24c ⟨Code 18a⟩+≡

```
;;; Code:
;;;; Compiler directives
(eval-when-compile (require 'wid-edit))

;;;; Internals
⟨Customization and global variables 1⟩
⟨Widgets 2⟩
⟨WHYSE project structure 4b⟩
⟨with-project 14d⟩
⟨buffer parsing function 15a⟩
⟨open-project-hook 6b⟩
⟨default hook functions 6c⟩

;;;; Commands
;;;###autoload
⟨WHYSE 3⟩
```

24d ⟨provide the whyse feature and list the file local variables 24d⟩≡

```
(provide 'whyse)

;; Local Variables:
;; mode: emacs-lisp
;; no-byte-compile: t
;; no-native-compile: t
;; End:
```

Appendix B

TESTING

TODO Adopt the ERT (Emacs Regression Tests) package to test WHYSE features as they are developed and become featureful. When a feature is implemented a test should be written which conforms to the current documentation so that regressions can be caught when changes are made.

TODO Adopt/use `makem.sh`, by "alphapapa".

B.1 Parsing tool syntax within a temporary buffer

25 ⟨test-parser-with-temporary-buffer.el 25⟩≡

```
;; -*- lexical-binding: nil; -*-
(defvar w--parse-success t
  "A simple boolean regarding the success or fialure of the last
  attempt to parse a buffer of Noweb tool syntax.")

⟨buffer parsing function 15a⟩
(with-temp-buffer
  (insert (shell-command-to-string
           "make --silent --file ~/src/whyse/Makefile tool-syntax"))
  (goto-char (point-min))
  (w--parse-current-buffer-with-rules))

;; Local Variables:
;; mode: lisp-interaction
;; no-byte-compile: t
;; no-native-compile: t
;; eval: (read-only-mode)
;; End:
```

Defines:
`w--parse-success`, used in chunk 15a.
Uses `w--parse-current-buffer-with-rules` 15a and `whyse` 1 3.

Appendix C

Indices

C.1 Chunks

⟨buffer parsing function 15a⟩ 15a, 24c, 25
⟨chunks and their boundaries 8b⟩ 7a, 8b, 9a, 10b, 10c
⟨Code 18a⟩ 18a, 20b, 23a, 24c, 28a, 28c, 29b
⟨collect child chunk uses 22⟩ 22
⟨Commentary 24b⟩ 23a, 24b
⟨create a database connection 5c⟩ 5b, 5c
⟨cross-referencing keywords 13⟩ 11a, 13
⟨Customization and global variables 1⟩ 1, 4a, 15b, 24c
⟨default hook functions 6c⟩ 6c, 19, 24c
⟨Emacs Lisp package headers 23b⟩ 23a, 23b
⟨error-causing keywords 14a⟩ 11a, 14a
⟨files and their paths 8a⟩ 7a, 8a
⟨find section title 21⟩ 20b, 21
⟨functions for navigating WHYSE parse trees 17⟩ 17, 18a
⟨functions to collapse text and newline tokens into their largest possible form 18b⟩ 18a, 18b
⟨fundamental indexing keywords, which are restricted to within a code chunk 11c⟩ 11a, 11c
⟨high-level Noweb tool syntax structure 7b⟩ 7a, 7b
⟨identifiers defined in a chunk 12a⟩ 11c, 12a
⟨identifiers used in a chunk 12b⟩ 11c, 12b
⟨index nl error message 14b⟩ 12d, 14b
⟨indexing and cross-referencing set-off words 11b⟩ 11a, 11b
⟨keyword definitions 10e⟩ 7a, 10e, 11a
⟨Licensing and copyright 24a⟩ 23a, 24a
⟨map over SQL s-expressions, creating the tables 6a⟩ 5b, 6a
⟨meta rules 7c⟩ 7a, 7c
⟨module-header keymap 28d⟩ 28c, 28d
⟨module-header top-level 28b⟩ 2, 28a, 28b, 29a
⟨module-header-center 29d⟩ 29b, 29d
⟨module-header-left 29c⟩ 29b, 29c
⟨module-header-right 29e⟩ 29b, 29e
⟨open-project-hook 6b⟩ 6b, 24c
⟨parse-project-in-temp-buffer 14c⟩ 3, 14c
⟨PEG rules 7a⟩ 7a, 15a
⟨provide the whyse feature and list the file local variables 24d⟩ 23a, 24d
⟨push the compiled SQL to the database and to the history stack 20a⟩ 20a
⟨quotations 10d⟩ 7a, 10d
⟨required packages 23d⟩ 23c, 23d
⟨return a filename for the project database 5a⟩ 5a, 5c
⟨setup project database 5b⟩ 3, 5b
⟨structural keywords 9c⟩ 9a, 9c
⟨structural keywords (except quotations) 9b⟩ 9b, 9c

⟨tagging keywords 9d⟩ 9a, 9d
⟨test-parser-with-temporary-buffer.el 25⟩ 25
⟨the index of identifiers 12c⟩ 11a, 12c
⟨tool errors 10a⟩ 9a, 10a
⟨unsupported indexing keywords 12d⟩ 11a, 12d
⟨WHYSE 3⟩ 3, 24c
⟨WHYSE project structure 4b⟩ 4b, 24c
⟨whyse-pkg.el 23c⟩ 23c
⟨whyse.el 23a⟩ 23a
⟨Widgets 2⟩ 2, 24c
⟨with-project 14d⟩ 14d, 24c

C.2 Identifiers

w--chunk-name: 17, 19
w--chunk-number: 17, 19
w--chunk-section: 19, 21
w--chunk-text: 17, 19
w--concatenate-text-tokens: 10b, 18b
w--get-chunk-uses-of-chunks: 19
w--log-in-buffer: 28a
w--nth-chunk-of-document: 17
w--nth-document: 17
w--nth-document-file-name: 17
w--parent-child-relationships: 19
w--parse-current-buffer-with-rules: 14c, 15a, 25
w--parse-failure-function: 15a
w--parse-success: 15a, 15b, 25
w--prepare-sexp-sql-from-file-tokens: 19, 19, 19
w--vectorize-chunk-data: 19
w-load-default-project?: 3, 4a
w-open-customize-when-no-projects-defined?: 3, 4a
w-open-project-hook: 6b, 6b
w-registered-projects: 1, 1, 3
w-with-project: 14d
whyse: 1, 1, 1, 1, 1, 3, 4a, 1, 1, 10c, 1, 23b, 23c, 1, 25

List of notes

1 CITE: an academic paper written in 1991 by Brown and Czejdo 1
2 TODO: Ensure that the previous statement in-prose [not in the TODO summary] is still correct. 4
3 TODO: Describe initialization of the system after parsing. 5
4 TODO: finish the creation of a database. Use what I learned in the fall! 5
5 CITE: Brown and Czejdo 6
6 CITE: Noweb Hacker's Guide 7
7 TODO: Verify that this statement is true: "Usually Noweb will warn a user that a chunk was referenced but undefined, or that there was some other issue with chunks." 10
8 TODO: Better document the functionality of the default hook function: `w--prepare-sexp-sql-from-file-tokens`. The current explanation is a copy of the brief explanation in the hook documentation string. 19
9 TODO: Make the number of entries in this history list customizable, limited to some given number or a default of one hundred. 20
10 TODO: Adopt the ERT (Emacs Regression Tests) package to test WHYSE features as they are developed and become featureful. When a feature is implemented a test should be written which conforms to the current documentation so that regressions can be caught when changes are made. 25
11 TODO: Adopt/use `makem.sh`, by "alphapapa". 25

C.3 Miscellaneous code and functions useful for development and debugging

28a ⟨Code 18a⟩+≡

```
(defun w--log-in-buffer (buffer-name &rest body)
  "In a new buffer named BUFFER-NAME, insert the value of evaluating BODY."
  (save-mark-and-excursion
    (with-current-buffer
        (generate-new-buffer buffer-name)
      (insert (format-message "%S" body)))))
```

⟨module-header top-level 28b⟩

Defines:
`w--log-in-buffer`, never used.

The top-level of the module header is defined here now.

Code modules have names, but documentation modules only sometimes have sections. Sometimes a documentation module doesn't have a section because the definition of one or more sections occur within it, and therefore a section cannot apply to it by reason of occurring before the beginning of the module. Therefore the headerbar must have a section within it for holding a program code module name or LaTeX section name. Additionally it must also have an area to contain further buttons. It makes sense to define, loosely, a left, right, and center area to be filled with other widgets (which will inherit widths of the area).

28b ⟨module-header top-level 28b⟩≡

```
:width (buffer-size)
:tag " "
:format "%t%v"
;; :keymap w-widget-headerbar-map
```

A keymap provides for handling keyboard and mouse events.

28c ⟨Code 18a⟩+≡
⟨module-header keymap 28d⟩

28d ⟨module-header keymap 28d⟩≡

29a ⟨module-header top-level 28b⟩+≡
```
:value '(module-header-left module-header-center module-header-right)
```

29b ⟨Code 18a⟩+≡
```
⟨module-header-left 29c⟩
⟨module-header-center 29d⟩
⟨module-header-right 29e⟩
```

29c ⟨module-header-left 29c⟩≡
```
(define-widget 'w--module-header-left 'group
  "DOCSTRING"
  (link :format "%t"
        :tag (format "<<%s>>%s%s%d"
                     module-name
                     addition?
                     modified?
                     module-number)))
```

29d ⟨module-header-center 29d⟩≡
```
(define-widget 'w--module-header-center 'group
  "DOCSTRING"
  '((item :tag "module")))
```

29e ⟨module-header-right 29e⟩≡
```
(define-widget 'w--module-header-right 'group
  "DOCSTRING"
  '((item :tag "module")))
```

Appendix D

License

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

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

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

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

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

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

Terms and Conditions

0. Definitions.

   “This License” refers to version 3 of the GNU General Public License.

   “Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   “The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

   To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

   A “covered work” means either the unmodified Program or a work based on the Program.

   To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

   A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

   The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

   The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make

modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

   (b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

(c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

(d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

(e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

    Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

    In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

    If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

    If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

    A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

    Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

    If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work,

but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

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

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

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

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

16. Limitation of Liability.

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

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

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

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

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
```

```
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

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

You should have received a copy of the GNU General Public License
along with this program.  If not, see <https://www.gnu.org/licenses/>.
```

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

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

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see *https://www.gnu.org/licenses/*.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read *https://www.gnu.org/licenses/why-not-lgpl.html*.