# 1 Projects

WEB Hypertext System's Emacs implementation (WHS) is a project-based application. Projects are lists registered with WHS using the "Easy Customization Interface", which provides a simple way to make the necessary information known to WHS. Users register a literate programming project (only Noweb–based programming is supported) as an item in the customization variable `whs-registered-projects`; further project data is contained in a Common Lisp struct during runtime.

In short, a project is composed of several things:

- a name,
- a Noweb source file,
- a shell command to run a user-defined script
- an SQLite3 database, and a connection thereto,
- a frame,
- and date-time information (creation, edition, and export).

The struct keeps some information during runtime, like the connection, but other information is generated at runtime (such as the filename of the database). These items are each explained in this section. If some item is not well-enough explained in this section, please try editing the Noweb source and improving the explanataion and creating a pull-request against the WHS Emacs Lisp repository on its Git forge; you may also submit your edition by email to the package maintainer.

Users of WHS in Emacs are expected to be familiar with Noweb; this does not include how Noweb is built from source (that is arcane, supposedly) or how filters are implemented with Sed, AWK, or other languages. Users must know, however, how to write a custom command-line for Noweave (read the manual section regarding the `-v` option).

Developers of WHS extensions (in either SQL or Emacs Lisp) 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.

A customization group for WHS 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⟩≡ (17b) 3c▷

```
(defgroup whs nil
  "The WEB Hypertext System."
  :tag "WHS"
  :group 'applications)

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

Defines:
`whs`, used in chunks 3–7 and 15.
`whs-registered-projects`, used in chunk 3.

The Widget feature is required by the registered projects variable, but may be redundant because the Easy Customization Interface is itself implemented with The Emacs Widget Library. Requiring the library may be undesireable, as `(require 'widget)` will be eagerly evaluated upon Emacs' initialization when `whs-registered-projects` is set to its saved custom value. However, there may be a good reason to eagerly evaluate that form: the Widget feature will be available immediately, and widgets will be used in buffers to provide TUI buttons for navigation between modules of a literate program (at least, that is the design of the program at this point in development), so having this feature available sooner than later is okay. The feature is required by the package regardless.

The `whs--project-widget` type used for the registered projects variable is a simple list widget containing the name of the project and its Noweb source file, along with a filename for a shell script 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 script is later executed by WHS upon loading the project, and the standard output captured for parsing by a PEG parser.

2 ⟨Widgets 2⟩≡ (17b)

```
(define-widget 'whs--project-widget 'list
  "The WHS 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 WHS for explanation."
           :value "")))
```

NB: Comments may be superfluous in a literate document like this, but some effort was made to produce a readable source file regardless of the general principles of literate programming; other authors write warnings into their tangled source files: “Don’t read this file! Read the Noweb source only!”. I don’t say that, especially for an Emacs application.

The sole interactive command—`whs`—loads the first element of `whs-registered-projects`, considering it the default project.

3a ⟨Quotation custom-set-variables 3a⟩≡
```
'(whs-registered-projects
  '(("Noweb Hypertext System"
     "~/Desktop/whs.nw"
     "make -C ~/Desktop --silent --file ~/src/whs/Makefile tool-syntax"))
  nil
  (widget))
```
Uses whs 1 3b and whs-registered-projects 1.

3b ⟨WHS 3b⟩≡ (17b)
```
(defun whs ()
  (interactive)
  (if-let ((whs-load-default-project?)
           (default-project (cl-first whs-registered-projects))
           (project (make-whs-project :name (nth 0 default-project)
                                      :noweb (nth 1 default-project)
                                      :script (nth 2 default-project))))
      ⟨System Initialization 4b⟩
      ⟨open Customize to register projects 3d⟩))
```
Defines:
whs, used in chunks 3–7 and 15.
Uses whs-load-default-project? 3c and whs-registered-projects 1.

WHS is likely to be useful for very large literate programs, so the command is designed to initialize from an existing project without prompt. In more verbose terms: unless `whs-load-default-project?` is non-nil and `whs-registered-projects` includes at least one element, Customize will be opened to customize the WHS group when `whs` is invoked.

3c ⟨Customization and global variables 1⟩+≡ (17b) ◁1
```
(defcustom whs-load-default-project? t
  "Non-nil values mean the system will load the default project.

nil will cause the interactive command `whs' to open Customize on
its group of variables."
  :type 'boolean
  :group 'whs
  :tag "Load default project when `whs' is invoked?")
```
Defines:
whs-load-default-project?, used in chunk 3.
Uses whs 1 3b.

3d ⟨open Customize to register projects 3d⟩≡ (3b)
```
(message "No WHS projects registered, or `whs-load-default-project?' is nil. %s"
         (customize-group 'whs))
```
Uses whs 1 3b and whs-load-default-project? 3c.

When `whs` is invoked, an instance of the project struct is created, and as a design goal is persisted using serialization after WHS exits.

4a ⟨WHS project structure 4a⟩≡ (17b)

```
(cl-defstruct whs-project
  "A WHS project"
  ;; Fundamental
  name
  noweb
  script
  database-file
  database-connection

  ;; Usage
  frame

  ;; Metadata
  (date-created (ts-now))
  date-last-edited
  date-last-exported

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

Uses `whs` 1 3b.

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 initalization (`frame`).

Initialization when the interactive command is called is covered next; to sumamrize: `whs-project-load-hook` is run.

## 2 System initialization from new projects

To summarize this section, since it is longer than the previous section, the object is the definition of ⟨System Initialization 4b⟩, which is a chunk used in `whs`.

In more explicit words, this section describes the actions that occur when a user invokes `whs` interactively (with M-x) and the preconditions have been met; the `whs` function has already been introduced, and only the "meaty" business end of its operation has been left undefined until now. Ergo, ⟨System Initialization 4b⟩ gathers together the functionality that converts a Noweb to its tool syntax with a project's specified shell script, sends the parsed text to the database, and finally creates the IDE frame.

4b ⟨System Initialization 4b⟩≡ (3b)

```
(let ((buffer (generate-new-buffer "WHS tool syntax generation shell output")))
  (with-current-buffer buffer
    ⟨run the project shell script to obtain the tool syntax 5a⟩

    ;; Go to the beginning of the buffer, then parse according to the PEG.
    (goto-char 0)
    ⟨parse the buffer with PEG rules 9a⟩))
```

## 2.1 Conversion to tool syntax

WHS could have been written to call the `noweave` programs itself, but that is less configurable than providing the opportunity to let the user configure this on their own. It respects Noweb's pipelines architecture, and keeps things as transparent as possible. What is needed to be Emacs Lisp is, and what is not isn't. The tool syntax is thus obtained by running the shell script configured for the project by calling it with the command-line provided in the third element of an entry in `whs-registered-projects`.

5a ⟨run the project shell script to obtain the tool syntax 5a⟩≡ (4b)

```
(make-process
 :name "whs-tool-generation"
 :buffer (get-buffer buffer)
 :command `("bash" ;; likely BASH on a GNU system, hoping for the `command-string' option.
            "-c"
            (,@(whs-project-script project)))
 :stderr (generate-new-buffer "WHS tool generation standard error stream")
 :sentinel (lambda (process event-string)
             (message "%S: %s" process event-string)))
```

Uses whs 1 3b.

The PEG for Noweb's tool syntax is run on the result of the shell script, and this value consumed by the parent of this chunk.

## 2.2 Database initialization

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 subdirectory 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.

5b ⟨return a filename for the project database 5b⟩≡ (5c)

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

Uses whs 1 3b.

For SQLite, the pathname 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.

5c ⟨create the database 5c⟩≡

```
(setf (whs-project-database-connection project)
      (emacsql-sqlite
       (whs-project-database-file ⟨return a filename for the project database 5b⟩)))
```

Uses whs 1 3b.

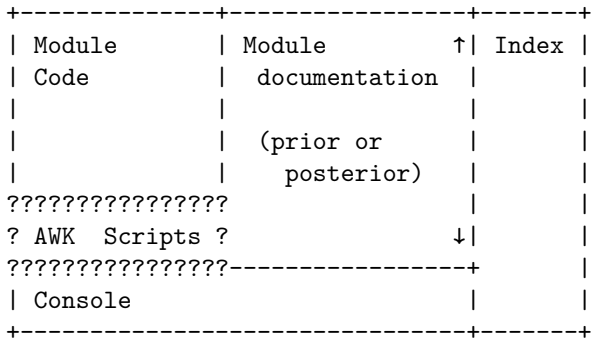


Figure 1: Simple drawing of WHS frame layout

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

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

```
(--map (emacsql (whs-project-database-connection project) it)

       ;; A list of SQL s-expressions to create the tables.
       '([:create-table module
          ([module-name
            content
            file-name
            section-name
            (displacement integer)
            (module-number integer :primary-key)])]

         [:create-table parent-child
          ([(parent integer)
            (child  integer)
            (line-number integer)]
           (:primary-key [parent
                          child]))]

         [:create-table identifier-used-in-module
          ([identifier-name
            (module-number integer)
            (line-number integer)
            type-of-usage]
           (:primary-key [identifier-name
                          module-number
                          line-number
                          type-of-usage]))]

         [:create-table topic-referenced-in-module
          ([(topic-name nil)
            (module-number integer)]
           (:primary-key [topic-name
                          module-number]))]))
```

Uses whs 1 3b.

### 2.3 Frame creation and atomic window specification

A frame like in Figure 2.3 should be created.

7a ⟨Get project frame 7a⟩≡
```
(progn
  (select-frame (whs-project-frame project))
  (switch-to-buffer (generate-new-buffer (whs-project-name project)) nil 'force-same-window)
  (let* ((window-right (split-window-right))
         (parent-window (window-parent window-right)))
    (window-make-atom parent-window)
    (display-buffer-in-atom-window
     (get-buffer-create (format "Module Index<%s>" (whs-project-name project)))
     `((window . ,parent-window) (window-height . 8)))))
```

Uses whs 1 3b.

## 3 System initialization from existing projects

WHS loads a project by running the shell script stored in the third element of the project list (which is pointed to by the script slot in the struct).

### 3.1 Initializing from an existing project

With a default project available, WHS runs whs-project-load-hook with the struct of the default project let-bound as project. Much of the functionality of WHS is implemented with the default hook, and extensions to WHS should be implemented by editing the WHS Noweb source and recompiling it, or extending the existing system with more hook functions added to the aforementioned hook list variable.

If the project's database file is empty (zero-bytes) or does not exist then the database is created from scratch. If the database already exists, the first module is loaded and the database is not changed.

7b ⟨delete the database if it already exists, but only if it's an empty file 7b⟩≡
```
;; Unless the SQLite database's size is zero or it doesn't exist, move it to the user's trash directory.
(let ((whs--dbfile (whs-project-database-file project)))
  (unless (or (not (file-exists-p whs--dbfile))
              (= 0 (file-attribute-size (file-attributes whs--dbfile))))

    ;; TODO: Is there a better way to do this? `backup-buffer'?
    (copy-file whs--dbfile (concat whs--dbfile "~") t)

    ;; TODO: ensure that this AREA of code is reasonable before release.
    ;; It may have been written to ease development only.
    (let ((delete-by-moving-to-trash t))
      (delete-file whs--dbfile t)))
```

Uses whs 1 3b.

# 4 Loading Noweb source files

To parse a noweb source file, the file needs to be loaded into a temporary buffer, then it can be parsed.

A simple usage of Noweb is given next, which shows that `noweave` does not include the header keyword, nor autodefinitions, usages, or indexing by default. Those are further stages in the UNIX pipeline defined by the user with `noweave` command-line program options and flags.

The WHS system parses the tool syntax emitted by `markup`, and early development versions (prior to version `0.n-devel`) completely ignore Noweb keywords out of that scope.

An example of a Noweb command-line a user may call is given next.

```
[bryce@fedora whs]$ noweave -v -autodefs elisp -index whs.nw 1>/dev/null
RCS version name $Name:  $
RCS id $Id: noweave.nw,v 1.7 2008/10/06 01:03:24 nr Exp $
(echo @header latex
/usr/local/lib/markup whs.nw
echo @trailer latex
) |
/usr/local/lib/autodefs.elisp |
/usr/local/lib/finduses |
/usr/local/lib/noidx |
/usr/local/lib/totex
```

Ergo, the simplified pipeline—using Emacs Lisp autodefinitions provided in Knoweb (written by Joseph S. Riel)—is as follows:

```
markup whs.nw | autodefs.elisp | finduses | noidx
```

### 4.0.1 In-development

For an existing project (during development, that is WHS) to be loaded, it must minimally be:

1. Parsed, then stored in a database
2. Navigable with WHS
   (a) Frame and Windows
   (b) Navigation buttons... at least for modules

This means diagramming the database schema, creating it in EmacSQL, creating validating functions for existing databases, exceptions for malformed databases, and documenting that in LaTeX.

Navigation with WHS is multi-part:

1. Query the database for a list of modules, and
2. Create a buffer for the text content retrieved

Exporting a project from the database and editing the project in an in-memory state are further objectives, but they will be achived after the above two have been implemented in a basic form.

### 4.0.2 TODO

The following features need to be implemented:

1. Project export from database to Noweb format
2. Editing of modules, documentation, and Awk code
3. Navigation with indices
4. Implement indices widgets

# 5 Parsing

This section covers the parsing of the Noweb tool syntax produced by a project shell script (described in §1). The following blocks of LISP code use the peg Emacs Lisp package to provide for automatic parser generation from a formal PEG grammar based off of the exhaustive description given in the Noweb Hacker's Guide.

Parsing the tool syntax allows for the generation of an partially–directed graph, a digraph, of the network of chunks which have hierarchical, self and non-self references, with their sequential ordering and non-sequential orderings available for navigation (see Intl. Soc. Knowledge Organization for futher information).

## 5.1 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.

9a ⟨parse the buffer with PEG rules 9a⟩≡ (4b)
```
;;;; Parsing expression grammar (PEG) rules
(with-peg-rules
    (⟨PEG rules 9b⟩)
  (peg-run (peg noweb)
           (lambda (lst)
             (message "Parsing failed in buffer:=%S.\nPEXes which failed:=%S" buffer lst))))
```

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

9b ⟨PEG rules 9b⟩≡ (9a)
```
⟨high-level Noweb tool syntax structure 9c⟩
⟨files and their paths 10a⟩
⟨chunks and their boundaries 10b⟩
⟨keywords 10c⟩
⟨meta rules 9d⟩
```

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 9c⟩ is defined.

9c ⟨high-level Noweb tool syntax structure 9c⟩≡ (9b)
```
;; Overall Noweb structure
(noweb (bob) (+ file) (if (eob))
       (action (message "End of buffer encountered while parsing.")))
```

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. The following ⟨meta rules 9d⟩ define rules which organize the constructs of a line-oriented, or data-oriented, syntax.

9d ⟨meta rules 9d⟩≡ (9b)
```
;; Helpers
(empty-line (bol) (eol) "\n")
(new-line (eol) "\n"
          (action (message "A new line was matched.")))
(not-eol (+ (not "\n") (any)))
(many-before-eol not-eol new-line)
```

With the ⟨meta rules 9d⟩ 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 is tangled when no specific root chunk is given on the command line.

10a ⟨files and their paths 10a⟩≡ (9b)
```
(file (bol) "@file" [space] (substring path) new-line
      (list (+ chunk))
      `(path chunk-list -- (list path chunk-list)))

(path (opt (or ".." ".")) (* path-component) file-name)
(path-component (and path-separator (+ [word])))
(path-separator ["\\/"])
(file-name (+ (or [word] ".")))
```

10b ⟨chunks and their boundaries 10b⟩≡ (9b)
```
(chunk begin (list (* keyword)) end)
(begin (bol) "@begin" [space] kind [space] ordinal (eol) "\n"
       (action (message "Chunk @begin matched.")))
(end (bol) "@end" [space] kind [space] ordinal (eol) "\n"
     (action (message "Chunk @end matched."))
     (opt (if begin)
          (action (message "[DEBUG] A begin [unconsumed] follows this end.")))
     `(kind-one ordinal-one keywords kind-two ordinal-two --
                (if (and (= ordinal-one ordinal-two) (string= kind-one kind-two))
                            ;;; Push the contents of the chunk to the stack in a cons
                            ;;; cell with the car being a list of the kind and number.
                            ;;;; E.g.:
                    ;; (("code" 3) . (@text @nl @text @nl))
                    (cons (list kind-one ordinal-one) keywords)
                  (error "There was an issue with unbalanced or improperly nested chunks."))))
(ordinal (substring [0-9] (* [0-9]))
         `(number -- (string-to-number number)))
(kind (substring (or "code" "docs")))
```

10c ⟨keywords 10c⟩≡ (9b)
```
(keyword
 (or
  ;; Keywords in the strict sense
  ⟨structural keywords 10d⟩
  ⟨tagging keywords 10e⟩

  ;; Keywords in the same strict sense, but which cause failures.
  ⟨usage errors 11a⟩
  ⟨tool errors 11b⟩))

⟨keyword definitions 12⟩
```

10d ⟨structural keywords 10d⟩≡ (10c)
```
;; structural
text
nl
defn
use ;; NOTE: related to the `identifier-used-in-module' table.
quotation
```

10e ⟨tagging keywords 10e⟩≡ (10c)
```
;; tagging
line
language
index
xref
```

11a ⟨usage errors 11a⟩≡ (10c)

```
;; user error
header
trailer
```

11b ⟨tool errors 11b⟩≡ (10c)

```
;; error
fatal
```

12 ⟨keyword definitions 12⟩≡ (10c)

```
(text (bol) "@text" [space] (substring (* (and (not "\n") (any)))) (eol) "\n")
(nl (bol) "@nl" (eol) "\n")

(defn "@defn" [space] (substring not-eol) new-line
  `(cdefn -- (cons "Chunk definition" cdefn)))

(use (bol) "@use" [space] (action (message "\"@use \" matched"))
     (substring not-eol) new-line
     `(chunk-name -- (if chunk-name
                         (cons "Chunk usage (child)" chunk-name)
                       (error "UH-OH! There's a syntax error in the tool output!"))))

(quotation beginquote (list (* keyword)) endquote)
(beginquote (bol) (substring "@quote") new-line)
(endquote (bol) (substring "@endquote") new-line
          `(bq kw eq -- (if (and (string= bq "@quote")
                                 (string= eq "@endquote"))
                            (cons "Quotation" kw)
                          (error "UH-OH! There's a parsing bug related to quotations."))))

(line (bol) "@line" [space] (substring ordinal) new-line
      `(o -- (cons "@line" o)))

(language (bol) "@language" [space] (substring many-before-eol))

;; NOTE: alike xref-keyword, index-keyword tokens handle the end of the
;; line regardless. The index token handles only the beginning of the
;; line.
(index (bol) "@index" [space] (opt index-keyword))

       ;;; FIXME: why did I define it with optionally? Is there a possibility
       ;;; that @xref can be followed by a newline directly?
(xref (bol) "@xref" [space]
      (action (message "\"@xref \" matched"))
      ;; NOTE: each xref-keyword individually handles the end of the
      ;; line, since it composes the remainder of the line regardless.
      (opt xref-keyword))

;; indexing keywords
(index-keyword
 (or
  i-defn
  i-localdefn
  i-use
  i-nl

  i-begindefs
  i-isused
  i-defitem
  i-enddefs

  i-beginuses
  i-isdefined
  i-useitem
  i-enduses

  i-beginindex
  i-entrybegin
```

```
  i-entryuse
  i-entrydefn
  i-entryend
  i-endindex)

 ;; IMPORTANT
 new-line)

(i-defn "defn" [space] (substring not-eol))
(i-localdefn "localdefn" [space] (substring not-eol))
(i-use "use" [space] (substring not-eol))
(i-nl "nl" [space] (substring not-eol))

(i-begindefs "begindefs")
(i-isused "isused" [space] (substring not-eol))
(i-defitem "defitem" [space] (substring not-eol))
(i-enddefs "enddefs")

(i-beginuses "beginuses")
(i-isdefined "isdefined" [space] (substring not-eol))
(i-useitem "useitem" [space] (substring not-eol))
(i-enduses "enduses")

(i-beginindex "beginindex")
(i-entrybegin "entrybegin" [space] (+ [word]) [space] (substring not-eol))
(i-entryuse "entryuse" [space] (substring not-eol))
(i-entrydefn "entrydefn" [space] (substring not-eol))
(i-endentry "entryend")
(i-endindex "endindex")

;; cross-referencing keywords
(xref-keyword
 (or
  x-label
  x-ref

  x-begindefs
  x-prevdef
  x-nextdef
  x-defitem
  x-enddefs

  x-beginuses
  x-useitem
  x-enduses
  x-notused

  x-beginchunks
  x-chunkbegin
  x-chunkuse
  x-chunkdefn
  x-chunkend
  x-endchunks

  x-tag)

 ;; IMPORTANT
 new-line
 (action (message "An xref-keyword was matched.")))
```

```
(x-label
 "label" [space] (substring not-eol))
(x-ref
 "ref" [space] (substring not-eol)
 `(reference -- (cons "XREF" reference)))

;; FIXME: TODO:
(x-begindefs (substring not-eol))

(x-prevdef
 "prevdef" [space] (substring not-eol))
(x-nextdef
 "nextdef" [space] (substring not-eol))

(x-beginuses
 "beginuses")
(x-useitem
 "useitem" [space] (substring not-eol))
(x-enduses
 "enduses")
(x-notused
 "notused" [space] (substring not-eol))

(x-beginchunks
 "beginindex")
(x-chunkbegin
 "chunkbegin" [space] (+ [word]) [space] not-eol)
(x-chunkuse
 "chunkuse" [space] (+ [word]))
(x-chunkdefn
 "chunkdefn" [space] (+ [word]))
(x-chunkend
 "chunkend")
(x-endchunks
 "endchunks")

;; Associates label with tag (word with not-eol)
(x-tag
 "tag" [space] (+ [word]) [space] not-eol)

;; User-errors (header and trailer) and tool-error (fatal)
(header (bol) "@header"
        (action (error "[ERROR] Do not use totex or tohtml in your noweave pipeline.")))
(trailer (bol) "@trailer"
         (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 repo
```

The result of parsing the beginning and end of a chunk are pushed onto the stack as a list.

The definition of a Noweb file, given by Ramsey, is simply a file containing one or more chunks; minimally, a Noweb file will contain the default documentation chunk.

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 push operations in `kind` and `ordinal` implement the beginning of a chunk, and the stack actions in the `end` rule check that the chunk elements on the stack are balanced.

The PEG library's stack operations may be a little confusing, so they will be discussed now.

Stack manipulating operations, such as `substring`, push their matching PEX input text to the stack. Labels in the left half of the stack actions pop elements off of the stack and let-bind them, making them easier to use in the right hand side. The right hand side uses let-bound variables and pushes its results back to the stack. Further, it is valid to pop elements of the stack without pushing them back (discarding them); it is also valid to push elements to the stack without popping anything off the stack.

Further rules needed to parse a chunk are an ordinal and the kind of chunk: code or documentation. An ordinal, in this sense, is any positive integer; it is a counting number. The kinds of chunks are represented by the strings "code" and "docs".

# 6 Packaging

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 Archvie). 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.

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

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

15a ⟨whs.el 15a⟩≡
```
⟨Emacs Lisp package headers 15b⟩
⟨Licensing and copyright 16b⟩
⟨Commentary 17a⟩
⟨Code 17b⟩
⟨EOF 17c⟩
```

15b ⟨Emacs Lisp package headers 15b⟩≡ (15a)
```
;;; whs.el --- WEB Hypertext System -*- lexical-binding: t -*-

;; Copyright © 2023 Bryce Carson

;; Author: Bryce Carson <bcars268@mtroyal.ca>
;; Created 2023-06-18
;; Keywords: tools tex hypermedia
;; URL: https://cyberscientist.ca/whs

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

15c ⟨whs-pkg.el 15c⟩≡
```
(define-package "whs" "0.1" "WEB Hypertext System"
  '(⟨required packages 16a⟩))
```
Uses `whs` 1 3b.

The Emacs Lisp Manual states, regarding the `Package-Requires` element of an Emacs Lisp package header:

> Its format is a list of lists on a single line.

Thus, to prevent spill–over in the printed document, the ⟨required packages 16a⟩ are given on separate lines in the literate document. When the file is tangled, however, a Noweb filter will be used to ensure that all required packages are on a single line by simply removing the new lines from the following code chunk. The same principle is followed for the ⟨file-local variables (never defined)⟩.

16a ⟨required packages 16a⟩≡ (15c)

```
(emacs "25.1")
(emacsql "20230220")
(dash "20230617")
(peg "1.0.1")
(cl-lib "1.0")
(ts "20220822")
```

16b ⟨Licensing and copyright 16b⟩≡ (15a)

```
;; 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/>.

;; If you cannot contact the author by electronic mail at the address
;; provided in the author field above, you may address mail to be
;; delivered to

;; Bryce Carson
;; Research Assistant
;; Dept. of Biology

;; Mount Royal University
;; 4825 Mount Royal Gate SW
;; Calgary, Alberta, Canada
;; T3E 6K6
```

17a ⟨Commentary 17a⟩≡ (15a)

```
;;; Commentary:
;; WHS 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 WHS, 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 WHS itself).
```

17b ⟨Code 17b⟩≡ (15a)

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

;;;; Internals
⟨Customization and global variables 1⟩
⟨Widgets 2⟩
⟨WHS project structure 4a⟩

;;;; Commands
;;;###autoload
⟨WHS 3b⟩
```

17c ⟨EOF 17c⟩≡ (15a)

```
(provide 'whs)

⟨file local variables 17d⟩
```

17d ⟨file local variables 17d⟩≡ (17c)

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

# 7 Indices

## 7.1 Chunks

⟨API-like functions 18⟩
⟨chunks and their boundaries 10b⟩
⟨Code 17b⟩
⟨Commentary 17a⟩
⟨create the database 5c⟩
⟨Customization and global variables 1⟩
⟨delete the database if it already exists, but only if it's an empty file 7b⟩
⟨Emacs Lisp package headers 15b⟩
⟨EOF 17c⟩
⟨file local variables 17d⟩
⟨files and their paths 10a⟩
⟨Get project frame 7a⟩
⟨high-level Noweb tool syntax structure 9c⟩
⟨keyword definitions 12⟩
⟨keywords 10c⟩
⟨Licensing and copyright 16b⟩
⟨map over SQL s-expressions, creating the tables 6⟩
⟨meta rules 9d⟩
⟨open Customize to register projects 3d⟩
⟨parse the buffer with PEG rules 9a⟩
⟨PEG rules 9b⟩
⟨Quotation custom-set-variables 3a⟩
⟨required packages 16a⟩
⟨return a filename for the project database 5b⟩
⟨run the project shell script to obtain the tool syntax 5a⟩
⟨structural keywords 10d⟩
⟨System Initialization 4b⟩
⟨tagging keywords 10e⟩
⟨tool errors 11b⟩
⟨usage errors 11a⟩
⟨WHS 3b⟩
⟨WHS project structure 4a⟩
⟨whs-pkg.el 15c⟩
⟨whs.el 15a⟩
⟨Widgets 2⟩

## 7.2 Identifiers

Underlined indices denote definitions; regular indices denote uses.



# 8 Appendices

## 8.1 A user-suggested functionality: `whs-with-project`

It was suggested during early development that ⟨API-like functions 18⟩ such as `whs-with-project` be written. An early version of such functionality is provided in `whs-with-project`.

18 ⟨API-like functions 18⟩≡
```
;; This chunk intentionally left blank at this time.
```

## 8.2 Testing the PEX grammar for Noweb's tool syntax

The following source text (given with LaTeX's verbatim block, rather than a Noweb code block) is a prior version of a working parse of Noweb's tool syntax.

```
@file /home/bryce/src/whs/whs.nw
@begin docs 0
@text The Bluetooth Device has been connected successfully.
@nl
@text The Bluetooth Device has been connected successfully.
@nl
@text The Bluetooth Device has been connected successfully.
@text The Bluetooth Device has been connected successfully.
@text The Bluetooth Device has been connected successfully.
@nl
@nl
@nl
@nl
@end docs 0
@begin docs 1
@text The Bluetooth Device has been connected successfully.
@nl
@end docs 1
@file ../whs.nw
@begin docs 0
@text Hello, world of literate programming.
@nl
@end docs 0

(defun test-pex ()
   (with-peg-rules
      ((noweb (+ file))
       (file (bol) "@file" [space] path (eol) "\n"
             (list (+ chunk))
             `(filename chunks -- (cons filename chunks)))

       ;;; Valid filenames the PEX was tested against.
       ;; /home/bryce/src/whs/whs.nw
       ;;; and,
       ;; ../whs.nw
       (path (substring (opt (or ".." ".")) (* path-component) file-name))
       (path-component (and path-separator (+ [word])))
       (path-separator ["\\/"])
       (file-name (+ (or [word] ".")))

       (begin (bol) "@begin" [space] kind [space] ordinal (eol) "\n")
       (end (bol) "@end" [space] kind [space] ordinal (eol) "\n"
            `(k1 z1 keywords k2 z2 --
                (if (and (= z1 z2) (string= k1 k2))
              ;;; Push the contents of the chunk to the stack in a cons
              ;;; cell with the car being a list of the kind and number.
              ;;;; E.g.:
                     ;; (("code" 3) . (@text @nl @text @nl))
                     (cons (list k1 z1) keywords)
                   (error "There was an issue with unbalanced or improperly nested chunks."))))
```

```
        (ordinal (substring [0-9] (* [0-9]))
                 `(number -- (string-to-number number)))
        (kind (substring (or "code" "docs")))

        ;; Chunk keyword definitions for what is in pextest.el (this file).
        (chunk begin
               ;; keywords
               (list (* (or text nl)))
               end)
        (text (bol) "@text" [space] (substring (* (and (not "\n") (any)))) (eol) "\n")
        (nl (bol) "@nl" (eol) "\n"))
    (goto-char 0)
    (peg-run (peg noweb)
             ;; (lambda (&rest args)
             ;;   (message "failure\n%s" args))
             ;; (lambda (&rest args)
             ;;   (message "success\n%S" args))
             )))

(test-pex)

;; Local Variables:
;; major-mode: lisp-interaction
;; End:
```