1 Preface

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 in the ⟨Commentary 17a⟩ for ideating WHS itself).

1.1 On literate programming

Literate programs can be organized in multiple ways; particularly, I note these forms of organization here. How WHS implementations may influence literate programming style, taste, or form will be interesting to observe (as it is a multi-langual art and will benefit from both the traditional language arts greatly as well as "code smell" [a strange term programmers have invented to somewhat describe computer–language arts]).

1. Algorithmic
2. Architectural
3. Linear
4. Notebook–like (Jupyter and iPython–like, which were influenced by Sweave)
5. Sweave–like (R Noweb ::= R Markdown)

The organization of this literate program is linear, with aspects of the program explained as the user would encounter them, more or less.

2 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.

2 ⟨Customization and global variables 2⟩≡ (17b) 4b▷

```
(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 4–8, 11, 16a, and 17c.
`whs-registered-projects`, used in chunk 4a.

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.

3 ⟨Widgets 3⟩≡ (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.

4a ⟨WHS 4a⟩≡ (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 5b⟩
      ⟨open Customize to register projects 4c⟩))
```

Defines:
  `whs`, used in chunks 4–8, 11, 16a, and 17c.
Uses `whs-load-default-project?` 4b and `whs-registered-projects` 2.

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.

4b ⟨Customization and global variables 2⟩+≡ (17b) ◁2
```
(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 4.
Uses `whs` 2 4a.

4c ⟨open Customize to register projects 4c⟩≡ (4a)
```
(message "No WHS projects registered, or `whs-load-default-project?' is nil. %s" (customize-group 'whs))
```
Uses `whs` 2 4a and `whs-load-default-project?` 4b.

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

5a ⟨WHS project structure 5a⟩≡ (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` 2 4a.

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.

3 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 5b⟩, 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 5b⟩ 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.

5b ⟨System Initialization 5b⟩≡ (4a)
```
(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 6a⟩

   ;; Go to the beginning of the buffer, then parse according to the PEG.
   (goto-char (point-min))
   ⟨parse the Noweb tool syntax with a PEG 10⟩))
```

3.1 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`.

6a ⟨run the project shell script to obtain the tool syntax 6a⟩≡ (5b)

```
(make-process
 :name "whs-tool-generation"
 :buffer (get-buffer buffer)
 :command `("sh" ;; 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 2 4a.

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.

3.2 Database initialization

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

6b ⟨return a filename for the project database 6b⟩≡ (6c)

```
(file-name-concat
 user-emacs-directory
 (when (f-directory? (expand-file-name ".cache" user-emacs-directory))
   ".cache")
 (concat (whs-project-name project)
         ".db"))
```

Uses whs 2 4a.

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.

6c ⟨create the database 6c⟩≡

```
(setf (whs-project-database-connection project)
      (emacsql-sqlite
       (whs-project-database-file ⟨return a filename for the project database 6b⟩)))
```

Uses whs 2 4a.

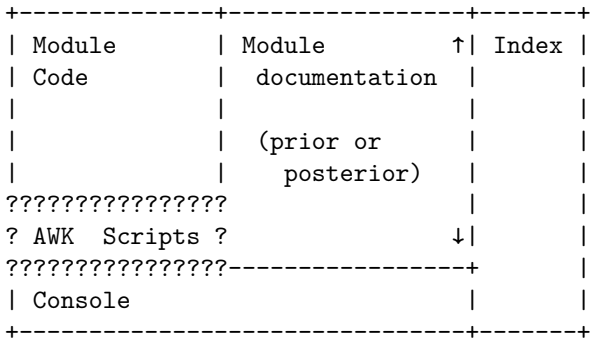

Figure 1: Simple drawing of WHS frame layout

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

7 ⟨map over SQL s-expressions, creating the tables 7⟩≡

```
(--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 2 4a.

3.3 Frame creation and atomic window specification

A frame like in Figure 3.3 should be created.

8a ⟨Get project frame 8a⟩≡
```
(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 2 4a.

4 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). The script is run by calling

4.1 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.

8b ⟨delete the database if it already exists, but only if it's an empty file 8b⟩≡
```
;; 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 2 4a.

5 Loading and Parsing Noweb source files

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 `noweave` command-line program's options and flags.

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

```
[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
```

5.0.1 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.

5.0.2 To do

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

6 Parsing

This section covers the actual parsing of the tool syntax. Production of the tool syntax from a Noweb project is covered in §2. The following lisp code uses the peg Emacs Lisp package to provide a parser that matches the lexed tokens and executes actions.

10 ⟨parse the Noweb tool syntax with a PEG 10⟩≡ (5b)

```
(peg-run #'peg-rule\ noweb
         (lambda (pegs)
           (while pegs
             (message "PEG failed: %S" (pop pegs)))))
```

6.1 PEG rules

It appears, unfortunately, that there are issues with the PEG I have defined, or with the library itself. With an expression like `(+ (not (null)))` or `(+ (not (eol)))`, the return value when these are wrapped with a substring stack action is `""`, an empty string. Using the rule `file` fails because it only matches `"@file "` for some reason, like that reason the return value is `""` in the previous clause.

11 ⟨PEG rules 11⟩≡ (17b)

```
;;;; Parsing expression grammar (PEG) rules
;; The `list' symbol in this rule, and all `peg' rules, is a
;; stack-action, not the normal Emacs Lisp `list' function symbol.
(define-peg-rule noweb
  () (bob) (list (+ file)) (* "\n") (eob)
  `(parse-tree
    -- (if parse-tree
           (progn (message "%S" parse-tree)
                  (with-temp-buffer
                    (insert parse-tree)
                    (write-file (expand-file-name "whs--parse-tree.el" user-emacs-directory)))
                  (unless (expand-file-name "whs--parse-tree.el" user-emacs-directory)
                    (error "Parse tree temporary file not written!"))
                  parse-tree)
         (error "`parse-tree' was nil!"))))

(define-peg-rule file
  () (bol) "@file" [space] (substring (+ (or multi-word-string ["." "\\" "/"]))) (eol) (+ chunk))

;;; NOTE: it would be helpful to construct this sort of parse tree at
;;; the CHUNK level, and this information can be directly sent to the
;;; database.
;; `((n . ,n)
;;   (name . ,substr)
;;   (offset . ,offset) ;; Displacement: count of @nl encountered in this file so far.
;;   (file . ,file)
;;   (section . ,section)) ;; Discover a single LaTeX sectioning command in
                         ;; the @text commands which are prior to this
                         ;; module's definition, as that is the direct
                         ;; parent section of this module.
;;; SQL attributes in the `module' table
;; module_number; module_name; displacement; file_name; section_name
(define-peg-rule chunk
  () (list begin (* keyword)) end)

;; what is the orientation of the stack?
(define-peg-rule begin
  () (bol) "@begin" [space] kind [space] zord (eol))

(define-peg-rule end
  () (bol) "@end" [space] kind [space] zord (eol)
  `(a b c d -- (progn (message "kind-equal? %S\nn-equal? %s"
                               (string= b d)
                               (= a c))
                      (message (cons a b))
                      (cons a b))))

(define-peg-rule ordinal ()
                 [1-9] (* [0-9]))

(define-peg-rule zord ()
```

```
    (substring (or "0" ordinal))
    `(number -- (string-to-number number)))

  (define-peg-rule kind ()
    (substring (or "code" "docs")))

  ;; FIXME: these need to be non-prioritized alternatives. Any one of
  ;; these keywords may be encountered, so they should be changed to
  ;; `opt'.
  (define-peg-rule keyword
    ()

    ;; structural
    (* text)
    (* nl)
    (* defn)
    (* use) ;; NOTE: related to the `identifier-used-in-module' table.
    (* quote)
    (* endquote)

    ;; tagging
    (* line)
    (opt language)
    (* index)
    (* xref)

     ;;; wrapping FIXME: these rules need to be defined, and they must
     ;;; NEVER be matched. If they match, the user has written a bad
     ;;; script, because totex or tohtml has been used in their noweave
     ;;; pipeline.
    (opt header)
    (opt trailer)

    ;; error
    (opt fatal))

  ;; strings containing spaces must match: "filename containing spaces.txt"
  (define-peg-rule multi-word-string
    () (+ [word]) (* (and [space] (+ [word]))))
  (define-peg-rule text
    () (bol) "@text" [space] (opt (any)) (eol))
  (define-peg-rule nl
    () (bol) "@nl" (eol))
  (define-peg-rule defn
    () (bol) "@defn" [space] (substring multi-word-string) (eol))
  (define-peg-rule use
    () (bol) "@use" [space] (any) (eol))
  (define-peg-rule quote
    () (bol) "@quote" (eol))
  (define-peg-rule endquote
    () (bol) "@endquote" (eol))

  (define-peg-rule line
    () (bol) "@line" [space] ordinal (eol))
  (define-peg-rule language
    () (bol) "@language" [space] (any) (eol))
  (define-peg-rule index
    () (bol)
    (or (and "@index" [space] index-keyword)
```

```
      "@index")
  (eol))
(define-peg-rule xref
  () (bol)
  (or (and "@xref" [space] xref-keyword)
      "@xref")
  (eol))

;; FIXME: these need to be non-prioritized alternatives. Any one of
;; these keywords may be encountered, so they should be changed to
;; `opt'.
;; indexing keywords
(define-peg-rule 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))

(define-peg-rule i-defn ()
                 "defn" [space] multi-word-string)
(define-peg-rule i-localdefn ()
                 "localdefn" [space] multi-word-string)
(define-peg-rule i-use ()
                 "use" [space] multi-word-string)
(define-peg-rule i-nl ()
                 "nl" [space] multi-word-string)

(define-peg-rule i-begindefs ()
                 "begindefs")
(define-peg-rule i-isused ()
                 "isused" [space] multi-word-string)
(define-peg-rule i-defitem ()
                 "defitem" [space] multi-word-string)
(define-peg-rule i-enddefs ()
                 "enddefs")

(define-peg-rule i-beginuses ()
                 "beginuses")
(define-peg-rule i-isdefined ()
                 "isdefined" [space] (+ [word]))
(define-peg-rule i-useitem ()
```

```
                  "useitem" [space] multi-word-string)
  (define-peg-rule i-enduses ()
                  "enduses")

  (define-peg-rule i-beginindex ()
                  "beginindex")
  (define-peg-rule i-entrybegin ()
                  "entrybegin" [space] (+ [word]) [space] multi-word-string)
  (define-peg-rule i-entryuse ()
                  "entryuse" [space] (+ [word]))
  (define-peg-rule i-entrydefn ()
                  "entrydefn" [space] (+ [word]))
  (define-peg-rule i-endentry ()
                  "entryend")
  (define-peg-rule i-endindex ()
                  "endindex")

  ;; FIXME: these need to be non-prioritized alternatives. Any one of
  ;; these keywords may be encountered, so they should be changed to
  ;; `opt'.
  ;; cross-referencing keywords
  (define-peg-rule xref-keyword ()
    (or
     x-label
     x-ref

     x-prevdef
     x-nextdef

     x-begindefs
     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))

  (define-peg-rule x-label ()
    "label" [space] (+ [word]))
  (define-peg-rule x-ref ()
    "ref" [space] (+ [word]))

  (define-peg-rule x-prevdef ()
    "prevdef" [space] (+ [word]))
  (define-peg-rule x-nextdef ()
    "nextdef" [space] (+ [word]))

  (define-peg-rule x-beginuses ()
    "beginuses")
```

```
(define-peg-rule x-useitem ()
  "useitem" [space] (+ [word]))
(define-peg-rule x-enduses ()
  "enduses")
(define-peg-rule x-notused ()
  "notused" [space] multi-word-string)

(define-peg-rule x-beginchunks ()
  "beginindex")
(define-peg-rule x-chunkbegin ()
  "chunkbegin" [space] (+ [word]) [space] multi-word-string)
(define-peg-rule x-chunkuse ()
  "chunkuse" [space] (+ [word]))
(define-peg-rule x-chunkdefn ()
  "chunkdefn" [space] (+ [word]))
(define-peg-rule x-chunkend ()
  "chunkend")
(define-peg-rule x-endchunks ()
  "endchunks")

;; Associates label with tag (word with multi-word-string)
(define-peg-rule x-tag ()
  "tag" [space] (+ [word]) [space] multi-word-string)

;; User-error (header trailer) and tool-error (fatal)
(define-peg-rule header
  () (bol) "@header" [space] (+ (not (null))) (eol)
  (action (error "User error. Do not use totex or tohtml in your noweave pipeline.")))
(define-peg-rule trailer
  () (bol) "@trailer" [space] (+ (not (null))) (eol)
  (action (error "User error. Do not use totex or tohtml in your noweave pipeline.")))
(define-peg-rule fatal
  () (bol) "@fatal" (* (any))
  (action (error "There was a fatal error in the pipeline. Debug the tools.")))
```

Uses whs 2 4a.

7 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.

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

16a ⟨Emacs Lisp package headers 16a⟩≡ (15)

```
;;; whs.el --- WEB Hypertext System -*- mode: emacs-lisp; lexical-binding: t; no-byte-compile: t; no-
native-compile: t; -*-

;; Copyright © 2023 Bryce Carson

;; Author: Bryce Carson <bcars268@mtroyal.ca>
;; Created 2023-06-18
;; Keywords: tools tex hypermedia
;; URL: https://cyberscientist.ca/whs
;; Package-Version: 0.1.0
;; Package-Requires: ((emacs "25.1") (emacsql "20230220") (dash "20230617") (peg "1.0.1") (cl-lib "1.0") (ts "202
```

Uses whs 2 4a.

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

```
;; 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⟩≡ (15)

```
;;; 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⟩≡ (15)

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

;;;; Internals
```

⟨Customization and global variables 2⟩
⟨Widgets 3⟩
⟨PEG rules 11⟩
⟨WHS project structure 5a⟩

```
;;;; Commands
;;;###autoload
```

⟨WHS 4a⟩

17c ⟨EOF 17c⟩≡ (15)

```
(provide 'whs)
;;; whs.el ends here
```

Uses whs 2 4a.

8 Indices

8.1 Chunks

8.2 Identifiers

Underlined indices denote definitions; regular indices denote uses.

9 Appendices

9.1 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.
```