

```

; -----
; Yet Another HyperTIES Implementation, This Time In Emacs (YAHTITTIE)
; Don Hopkins
; -----

; -----
; Notes:
;
; * Make synonyms use the abbrev mechanism
;   use-abbrev-table define-local-abbrev abbrev-mode abbrev-expansion
; * Edit definitions, etc, with edit-in-transient-window
; * Show temporary messages with typeout-text
; * Use (temp-use-buffer ...) instead of (s-e (s-t-b) ...)
; * Use (progn-args 1) for conditional execution of args
; * Use define-hooked-local-abbrev to make .commands prompt for
;   arguments, with completion over known names, in storyboard mode.
;   Look at /usr/unimacs/lib/emacs/mac/lib/cmacs.ml
;   When you type in a command like .target, a template is put down,
;   .target <target> <article>
;   and you're prompted for the arguments, which replace the
;   respective parts of the template. When prompting, it offers
;   help and completion over the appropriate name spaces on the
;   stack of master-indices.

; -----
; Description:
;
; Emacs reads in and parses the master index, a text file, which defines
; three name spaces:  articles, pictures, and targets.  The master-index
; maps titles and synonyms to entries.  Each master-index has three
; dictionaries of entries (article-dict, picture-dict, and target-dict),
; which map names to entry indices, which are used to index several
; parallel arrays containing information about the entries, including
; entry-titles (their titles -- not synonyms), entry-dicts (the name of
; one of the master index's dict buffer, in which the entry was defined,
; i.e. "master-index:articles"), entry-filenames (absolutized relative
; to the entry's master-index's directory), and entry-buffers (the
; buffer names of instantiated entries, of "" if uninstantiated).
; Entries are not instantiated until they are needed.  To instantiate an
; entry, its file is read into a buffer (invoking whatever major mode
; hooks are called for by the file name extension, i.e. entering
; storyboard mode for .st0 files), and then some buffer local variables
; are set up as its instance variables.  After that, some mode-specific
; setup function should be called.  <Implement this!>
;
; The buffer local entry instance variables include entry-index (its
; index in the arrays), entry-class (its class as determined by
; entry-filenames), field-names (an array of the names of fields that
; have been parsed out of the entry), &fields (the count of fields
; parsed so far), field-tops, and field-bottoms (arrays of marks of the
; field regions).  Other entry related variables are stored in the
; arrays, redundantly (entry-filenames, entry-buffers) or not
; (entry-titles, entry-dicts), or stored in buffer local variables of
; the dictionary buffers named in entry-dicts (dict-owner (the buffer
; name of the dict's master-index), dict-space (the name space of the
; dict:  "article", "picture", or "target")).
;
; When a named field of an entry is requested, the entry buffer local

```

```
; array field-names is searched for that name, and if it's found, the
; corresponding marks in field-tops and field-bottom are used to locate
; the field in the entry buffer. Otherwise, a mode-specific function is
; called to search for the field. If it finds the field, it should
; return 1 with the region around it. If the field can't be found, it
; should either creates a new empty field, returning 1 with the region
; around it, or give up and return 0 without creating a new field,
; according to some argument it's passed. If it returns 1, the newly
; found field is recorded in the entry buffer local field arrays, so
; it can be found very quickly next time.
;
```

```
; -----
; TODO:
; - function to create new database
```

```
; -----
; Info search list (for the menus)
```

```
(declare-global info-search-list-initialized)
(if (! info-search-list-initialized)
  (progn
    (move-to-head-of-search-list "info" "/tuntum/don/databases/info")
    (setq info-search-list-initialized 1)
  )
)
```

```
; -----
; Array size defaults:
```

```
; #master-indices      Initial master index stack size
(declare-global #master-indices)
(setq-default #master-indices 10)
```

```
; #entries             Initial entry array size
(declare-global #entries)
(setq-default #entries 500)
```

```
; #fields              Initial field array size
(declare-global #fields)
(setq-default #fields 5)
```

```
; #commands            Initial command array size
(declare-global #commands)
(setq-default #commands 20)
```

```
; #args                Initial command argument array size
(declare-global #args)
(setq-default #args 2)
```

```
; -----
; Global variables
```

```
; Use the same zero length array everywhere (saves memory)
(declare-global nullarray)
(setq-default nullarray (new-array 0))
```

```

; Should we eat a blank after eating a token?
(declare-global eat-blank?)
(setq-default eat-blank? 0)

; Should we cache field markers?
(declare-global cache-field-markers?)
(setq-default cache-field-markers? 0)

(declare-global cties-dir)
(setq-default cties-dir "/tuntum/guest/cties/")
;(setq-default cties-dir "/bensun/guest/cties/")

(declare-global canvas-editor)
(setq-default picture-editor
  (concat cties-dir "/bin/paintdemo")
)

(declare-global target-editor)
(setq-default target-editor
  (concat "psh " cties-dir "/pslib/postscript/animatoe.ps ; "
    "psh " cties-dir "/pslib/postscript/targettool.ps "
  )
)

; -----
; Global arrays:

;   &master-indices           Master index stack pointer
(declare-global &master-indices)

;   master-indices            Array of master index buffer names
(declare-global master-indices)

;   &entries                  Entry count
(declare-global &entries)

;   entry-titles              Array of entry titles
(declare-global entry-titles)

;   entry-dicts               Array of master index dict buffer names
(declare-global entry-dicts)

;   entry-filenames           Array of entry file names
(declare-global entry-filenames)

;   entry-buffers             Array of instantiated entry buffer names
(declare-global entry-buffers)

;   &commands                 Command count
(declare-global &commands)

;   command-names             Array of command names
(declare-global command-names)

;   command-spaces            Array of array of command argument space names
(declare-global command-spaces)

;   command-actions           Array of array of command argument actions

```

```

(declare-global command-actions)

; -----
; HyperTIES authoring mode

(defun (init-hyperties
  (save-excursion
    (save-window-excursion
      (init-menus)
      (setq &master-indices 0)
      (setq master-indices (ensure-array master-indices #master-indices))
      (setq &entries 0)
      (setq entry-titles (ensure-array entry-titles #entries))
      (setq entry-dicts (ensure-array entry-dicts #entries))
      (setq entry-filenames (ensure-array entry-filenames #entries))
      (setq entry-buffers (ensure-array entry-buffers #entries))
      (setq &commands 0)
      (setq command-names (ensure-array command-names #commands))
      (setq command-spaces (ensure-array command-spaces #commands))
      (setq command-actions (ensure-array command-actions #commands))
    )
  )
))

(declare-global hyperties-menus-loaded)

(defun (init-menus &info-exists
  ; main emacs menus loaded yet? (heuristic)
  (if (! hyperties-menus-loaded) ; hyperties menus loaded yet?
    (progn
      (message-for 1 "Loading menus...")
      (if (! (is-bound #menu-120))
        (error-occurred
          (ps-autoload "menuclasses.ps" "menuclasses.ps")
          (load-menu-tree "emacs") ; don't show the menu after loading it
        )
      )
      (error-occurred
        (load-menu-tree "hyperties")
        (load-menu-tree "hyperties-picture")
        (load-menu-tree "hyperties-target")
      )
      (setq hyperties-menus-loaded 1)
    )
  )
))

; -----
; Master index interface:

; Master index buffer specific variables:

; master-index-initialized Master index buffer initialization flag.
(declare-buffer-specific master-index-initialized)
(setq-default master-index-initialized 0)

; master-index-dir Directory name of this master index
(declare-buffer-specific master-index-dir)

```

```

; article-dict          Name of article dictionary buffer
(declare-buffer-specific article-dict)

; picture-dict          Name of picture dictionary buffer
(declare-buffer-specific picture-dict)

; target-dict           Name of target dictionary buffer
(declare-buffer-specific target-dict)

; Initialize the locals of the master index in the current buffer:
(defun (init-master-index
  (setq master-index-initialized 1)
  (setq master-index-dir
    (file-directory-name (buffer-file-name (current-buffer-name))))
  (setq article-dict (make-dict "article"))
  (setq picture-dict (make-dict "picture"))
  (setq target-dict (make-dict "target"))

  (instantiate-master-index)
))

; Master index instantiation:
(defun (instantiate-master-index
  $the-title $the-dict $the-filename
  (setq case-fold-search 1)
  (save-restriction
    (chomp-space "article"
      (put-in-dict $the-dict $the-title &entries))
    (chomp-space "picture"
      (put-in-dict $the-dict $the-title &entries))
    (chomp-space "target"
      (put-in-dict $the-dict $the-title &entries))
  )
))

; Chomp in a space from a master index:
(defun (chomp-space
  (narrow-to-sub-index (arg 1))
  (setq $the-dict (concat (current-buffer-name) ":" (arg 1)))
  (pop-up-message (concat "BEGIN chomp-space " $the-dict))
  (message-for 1 $the-dict)
  (setq $the-title "*bogus title*")
  (setq $the-filename "*bogus filename*")
  (while (! (error-occurred (search-forward "\"")))
    (set-mark)
    (search-forward "\"")
    (backward-character)
;    (setq $the-title (canonicalize (region-to-string)))
    (setq $the-title (region-to-string))
    (forward-character)
    (if (! (looking-at "\n"))
      (progn
        (skip-blanks-forward)
        (set-mark)
        (end-of-line)
        (setq $the-filename (region-to-string))
        (define-entry $the-dict $the-title $the-filename ""))
    )
  )

```

```

    )
  )
  (progn-args 2)
)
(pop-up-message (concat "END chomp-space " $the-dict))
(message-for 1 "end " $the-dict)
))

; Narrow the region to part of a master index describing the named space:
(defun (narrow-to-sub-index $name
  (setq $name (arg 1 ": narrow-to-sub-index (name) "))
  (widen-region)
  (beginning-of-file)

  (if (error-occurred
      (search-forward
        (concat "----- " $name "S -----\\n")))
    (progn ; make new sub-index & narrow to it
      (end-of-file)
      (set-mark)
      (insert-string "\\n----- " $name "s -----\\n")
      (case-region-upper)
    )
  )
  (set-mark)
  (if (error-occurred (search-forward "\\n----- "))
    (end-of-file)
    (beginning-of-line)
  )
  (narrow-region)
  (beginning-of-file)
))

; -----
; Index manager interface:

; Flag to tell if the master index stack has changed and we need to rebuild
; the completion list.
(declare-global master-index-stack-changed)
(setq-default master-index-stack-changed 1)

; Push master index buffer on stack:
; (push-master-index "master-index")
(defun (push-master-index
  (save-excursion
    (save-window-excursion
      (switch-to-buffer (arg 1 ": push-master-index (buffer) "))
      ; initialize master index buffer locals if necessary
      (if (! master-index-initialized)
        (init-master-index)
      )
      (if (<= (array-size master-indices) &master-indices)
        (setq master-indices (grow-array master-indices))
      )
      (setq-array master-indices (++ &master-indices) (current-buffer-name))

      (setq master-index-stack-changed 1)
    )
  )
)

```

```

)
))

(defun (push-master-index-file $pmif-file $pmif-buffer
  (if (interactive)
    (setq $pmif-file (get-tty-file ": push-master-index-file "))
    (setq $pmif-file (arg 1)))
  )
  (save-excursion
    (save-window-excursion
      (find-file $pmif-file)
      (setq $pmif-buffer (current-buffer-name))
    )
  )
  (push-master-index $pmif-buffer)
))

; Pop top master index from stack:
; (pop-master-index)
; Pop named master index from stack:
; (pop-master-index "master-index")
(defun (pop-master-index $name $i
  (if (<= (nargs) 0)
    (if (> &master-indices 0)
      (progn (-- &master-indices)
        (setq master-index-stack-changed 1)
      )
    )
    (progn
      (setq $name (arg 1 ": pop-master-index "))
      (if (= $name "")
        (pop-master-index) ; recurse
        (progn
          (setq $i (+ &master-indices 1))
          (while (> (-- $i) 0)
            (if (| (= $name
              (array master-indices $i))
              (= $name
              (buffer-file-name
                (array master-indices $i))))
              (progn ; Found it!
                (while (<= (++ $i) &master-indices)
                  (setq-array master-indices (- $i 1)
                    (array master-indices $i)))
                (-- &master-indices)
                (setq master-index-stack-changed 1)
                (setq $i 0)
              )
            )
          )
        )
      )
    )
  )
)

(defun (with-top-master-index
  (save-window-excursion

```

```

    (temp-use-buffer (array master-indices &master-indices)
      (progn-args 1)
    )
  )
))

; Make buffers of the sorted names of all the article, picture, and target
; names on the master index stack, to be used for completion and help.
(defun (update-completion-list
  (if master-index-stack-changed
    (progn
      (setq master-index-stack-changed 0)
      (make-completions "article")
      (make-completions "picture")
      (make-completions "target")
    )
  )
)
))

(defun (make-completions $mc-space $mc-i
  (setq $mc-space (arg 1 ": make-completions (space) "))
  (temp-use-buffer (concat "*" $mc-space "-completions*"))
  (erase-buffer)
  (setq $mc-i (+ &master-indices 1))
  (while (> (-- $mc-i) 0)
    (set-mark)
    (yank-buffer (concat (array master-indices $mc-i) ":" $mc-space))
    (goto-character (mark))
    (while (looking-at "\n") (delete-next-character))
    (end-of-file)
  )
  (set-mark)
  (beginning-of-file)
; (filter-region "sort")
)
))

; (get-space-word "Article: " "article")
(defun (get-space-word $gsdw-prompt $gsdw-buf
  (setq $gsdw-prompt (arg 1 ":_prompt: "))
  (setq $gsdw-buf (arg 2 ":_space: "))
  (setq $gsdw-buf (concat "*" $gsdw-buf "-completions*"))
  (get-tty-string $gsdw-prompt)
; (get-tty-word $gsdw-prompt $gsdw-buf)
)
))

(defun (foo
  (pop-up-message (get-default-space-word "Article: " "The Blue Whale" "article")))
)

; (get-default-space-word "Article: " "The Blue Whale" "article")
(defun (get-default-space-word $gdsw-prompt $gdsw-default $gdsw-space
  $gdsw-result
  (setq $gdsw-prompt (arg 1 ":_prompt: "))
  (setq $gdsw-default (arg 2 ":_default: "))
  (setq $gdsw-space (arg 3 ":_space: "))
  (if (!= $gdsw-default "")
    (setq $gdsw-prompt

```



```

        (concat $gdsw-prompt "(" $gdsw-default ") ")
    )
    (if (= (setq $gdsw-result (get-space-word $gdsw-prompt $gdsw-space)) "")
        $gdsw-default
        $gdsw-result
    )
))

; Returns index of entry in the global entry-* arrays, or 0 if not found:
; (search-master-indices "the blue whale" "picture") => entry-index | 0
(defun (search-master-indices $title $space
    $i $names $indices &last $j &entry $entry-buffer
    (save-excursion
        (save-window-excursion
            (setq $title (canonicalize (arg 1 ": search-master-indices (title) ")))
            (setq $space (arg 2 ": search-master-indices (space) "))
            (setq &entry 0)
            (setq $entry-buffer "")
            (setq $i (+ &master-indices 1))
            (while (> (-- $i) 0)
                (switch-to-buffer (array master-indices $i))
                (if (setq &entry (search-master-index $space $title))
                    (setq $i 0) ; exit loop if found
                )
            )
            &entry
        )
    )
))

; Search the master index in the current buffer, and return the entry
; index if found, or 0 if not.
; (search-master-index "article" "the blue whale")
(defun (search-master-index
    ; convert to number (expects numeric index!)
    (+ (get-from-dict (concat (current-buffer-name) ":" (arg 1)) (arg 2)))
))

; -----
; Entry interface:

; Entry buffer specific variables:

;   entry-index          Index of entry
(declare-buffer-specific entry-index)

;   entry-frame          Frame of entry
(declare-buffer-specific entry-frame)

;   entry-class          Class of entry
(declare-buffer-specific entry-class)
(setq-default entry-class "*bogus class*")

;   entry-field          Current entry field
(declare-buffer-specific entry-field)
(setq-default entry-field "")

;   &fields              Field count

```

```

(declare-buffer-specific &fields)

; field-names          Array of field names
(declare-buffer-specific field-names)
; field-tops           Array of field top marks
(declare-buffer-specific field-tops)
; field-bottoms        Array of field bottom marks
(declare-buffer-specific field-bottoms)

; entry-field-buffer    Name of buffer with field index abbrev map.
(declare-buffer-specific entry-field-buffer)

; entry-instantiation-hook  Name of function to call to instantiate entry,
;                           set up by major mode of entry file.
(declare-buffer-specific entry-instantiation-hook)
(setq-default entry-instantiation-hook "*bogus entry instantiation hook*")

; entry-author-hook     Name of function to call to start editing
;                           entry, set up by major mode of entry file.
(declare-buffer-specific entry-author-hook)
(setq-default entry-author-hook "*bogus entry author hook*")

; entry-find-field-hook  Name of function to call to find field in
;                           entry, set up by major mode of entry file.
(declare-buffer-specific entry-find-field-hook)
(setq-default entry-find-field-hook "*bogus entry find field hook*")

; Entry instantiation:
; Initialize entry instance variables, as locals in the current entry buffer.
; First argument is entry index:
; (instantiate-entry 13)
(defun (instantiate-entry

  (setq entry-index (arg 1 ": instantiate-entry (index) "))
  (setq entry-frame (current-frame))
  (setq entry-field "") ; Defaults to whole buffer
  (setq &fields 0)
  (setq field-names (new-array #fields))
  (setq field-tops (new-array #fields))
  (setq field-bottoms (new-array #fields))

  ; Make abbrev table to store field indices in
  (setq entry-field-buffer (concat (current-buffer-name) ":fields"))
  (temp-use-buffer entry-field-buffer
    (use-abbrev-table (current-buffer-name))
  )

  (execute-extended-command entry-instantiation-hook)

))

(defun (author-entry
  (progn (switch-to-frame entry-frame))
; (frame-to-top entry-frame)
  (narrow-to-field entry-field)
  (set-frame-label (get-entry-title entry-index))
  (set-icon-label (get-entry-title entry-index))
  (save-command-context)

```

```

    (execute-extended-command entry-author-hook)
  ))

; Returns the file name of an entry:
; (get-entry-filename (search-master-indices "mona lisa" "picture"))
;   => "/tumentum/don/db/mona-lisa.pn0"
(defun (get-entry-filename
  (array entry-filenames (arg 1))
))

; Returns the title of an entry:
; (get-entry-title (search-master-indices "mona" "article"))
;   => "Mona Lisa"
(defun (get-entry-title
  (array entry-titles (arg 1))
))

; Returns the dictionary buffer name where an entry is defined:
; (get-entry-dict (search-master-indices "mona lisa" "picture"))
;   => "master-index<2>:picture"
(defun (get-entry-dict
  (array entry-dicts (arg 1))
))

; Returns buffer name of entry, instantiating it if necessary:
; (get-entry-buffer (search-master-indices "mona lisa" "picture"))
;   => "mona-lisa.pn0"
(defun (get-entry-buffer &geb-index $geb-buffer
  (setq &geb-index (arg 1 ": get-entry-buffer (index) "))
  (if (= (setq $geb-buffer (array entry-buffers &geb-index)) "")
    (save-excursion
      (save-window-excursion
        ; invokes major mode setup func, setting entry-instantiation-hook
        (find-file (array entry-filenames &geb-index))
        (setq $geb-buffer (current-buffer-name))
        (setq-array entry-buffers &geb-index $geb-buffer)
        ; calls entry-instantiation-hook
        (instantiate-entry &geb-index)
        (author-entry)
      )
    )
  )
  $geb-buffer
))

; Return buffer name of entry's master index:
; (get-entry-owner entry-index) => "master-index<2>"
(defun (get-entry-owner $geo-dict
  (save-window-excursion
    (temp-use-buffer (get-entry-dict (arg 1))
      (setq $geo-dict dict-owner)
    )
  )
  $geo-dict
))

; Return directory name of entry's file:
; (get-entry-directory entry-index) => "/tumentum/don/db/"

```

```

(defun (get-entry-directory
  (file-directory-name (get-entry-filename (arg 1)))
))

; Return directory name of entry's master index:
; (get-entry-owner-directory entry-index) => "/tumtum/don/db/master-index"
(defun (get-entry-owner-directory $geod-dict
  (setq $geod-dict (arg 1))
  (save-window-excursion
    (temp-use-buffer (get-entry-dict $geod-dict)
      (switch-to-buffer dict-owner
        (setq $geod-dict master-index-dir)
      )
    )
  )
  $geod-dict
))

; Return name of entry's space (article, picture, target):
; (get-entry-space entry-index) => "article"
(defun (get-entry-space $ges-space
  (temp-use-buffer (get-entry-dict (arg 1))
    (setq $ges-space dict-space))
  $ges-space
))

; -----
; Dict stuff:
;
; Dict buffer specific variables:

; dict-owner      Buffer name of master index that owns this dict
(declare-buffer-specific dict-owner)
(setq-default dict-owner "*bogus owner*")

; dict-space      Name space of this dictionary (article, picture, target)
(declare-buffer-specific dict-space)
(setq-default dict-owner "*bogus space*")

; Takes the name of a dictionary space to create, whose owner is the
; current master index buffer, and returns the new dictionary buffer name:
; (make-dict "article") => "master-index<2>:article"
(defun (make-dict $md-space $md-buffer $md-owner

  (setq $md-space (arg 1 ": make-dict (space) "))
  (setq $md-owner (current-buffer-name))
  (setq $md-buffer (concat $md-owner ":" $md-space))

  (temp-use-buffer $md-buffer
    (setq needs-checkpointing 0)
    (erase-buffer)
    (insert-string "\n") ; delimiter to make searches easy

    ; The dictionary mappings are stored in the dictionary buffer's
    ; local abbrev table, of the same name as the buffer.
    (use-abbrev-table $md-buffer)
    (setq abbrev-mode 0) ; We don't want abbrevs in the buffer to be expanded
  )
)

```

```

    (setq dict-owner $md-owner)
    (setq dict-space $md-space)
  )
  $md-buffer ; return new dictionary buffer name
))

; Put a value into a dictionary:
; (put-in-dict "master-index<2>:articles" "!home" "1")
(defun (put-in-dict $dict-buffer $dict-key $dict-value
  (setq $dict-buffer (arg 1 ": put-in-dict (buffer) "))
  (setq $dict-key (case-string-lower (arg 2 ": put-in-dict (key) ")))
  (setq $dict-value (arg 3 ": put-in-dict (value) "))
  (temp-use-buffer $dict-buffer
    (beginning-of-file)
    (if (error-occurred
        (search-forward (concat "\n" $dict-key "\n")))
      (progn (end-of-file)
              (insert-string $dict-key "\n"))
      )
    )
  (define-local-abbrev $dict-key $dict-value)
)
))

; Retrieve a value from a dictionary:
; (get-from-dict "master-index<2>:articles" "!home") => "1"
(defun (get-from-dict $dict-buffer $dict-key
  (setq $dict-buffer (arg 1 ": get-from-dict (buffer) "))
  (setq $dict-key (arg 2 ": get-from-dict (key) "))
  (temp-use-buffer $dict-buffer
    (abbrev-expansion $dict-key)
  )
)
))

(defun (remove-from-dict $dict-buffer $dict-index $dict-key
  (setq $dict-buffer (arg 1 ": remove-from-dict (buffer) "))
  (setq $dict-index (arg 2 ": remove-from-dict (index) "))
  (temp-use-buffer $dict-buffer
    (beginning-of-file)
    (next-line)
    (beginning-of-line)
    (while (! (eobp))
      (set-mark)
      (end-of-line)
      (setq $dict-key (region-to-string))
      (error-occurred (forward-character))
      (if (= (abbrev-expansion $dict-key) $dict-index)
        (progn
          (define-local-abbrev $dict-key 0)
          (erase-region)
        )
      )
    )
  )
)
))

; -----
; Entry stuff:

```

```

;

; Makes the uninstantiated part of a new entry.
; Assumes we're in the master-index buffer.
(defun (define-entry $de-dict $de-title $de-filename $de-buffer
  (setq $de-dict (arg 1 ": define-entry (dict) "))
  (setq $de-title (arg 2 ": define-entry (title) "))
  (setq $de-filename (arg 3 ": define-entry (filename) "))
  (setq $de-buffer (arg 4 ": define-entry (buffer) "))
  (if (!= (substr $de-filename 1 1) "/")
      (setq $de-filename (concat master-index-dir $de-filename)))
  )
  (++) &entries)
  (if (< (array-size entry-titles) &entries)
      (setq entry-titles (grow-array entry-titles)))
  )
  (setq-array entry-titles &entries $de-title)
  (if (< (array-size entry-dicts) &entries)
      (setq entry-dicts (grow-array entry-dicts)))
  )
  (setq-array entry-dicts &entries $de-dict)
  (if (< (array-size entry-filenames) &entries)
      (setq entry-filenames (grow-array entry-filenames)))
  )
  (setq-array entry-filenames &entries $de-filename)
  (if (< (array-size entry-buffers) &entries)
      (setq entry-buffers (grow-array entry-buffers)))
  )
  (setq-array entry-buffers &entries $de-buffer)
))

(defun (forget-entry $fe-index $fe-buf
  (setq $fe-index (arg 1 ": forget-entry (index) "))
  (if (!= (setq $fe-buf (get-entry-buffer $fe-index)) "")
      (progn (delete-buffer $fe-buf)
              (setq-array entry-buffers $fe-index ""))
      )
  )
  )
))

; Relativize the file name following the dot in the master-index
; buffer.
(defun (relativize-file-name $rfn-name $rfn-dir
  (if (looking-at master-index-dir)
      (progn (set-mark)
              (goto-character (+ (dot) (length master-index-dir)))
              (erase-region)
              )
      )
  (if (! (looking-at "/"))
      (insert-string "./")
      )
  )
  )
))

(defun (new-entry $ne-space $ne-title $ne-file $ne-dict
  (setq $ne-space (arg 1 ": new-entry (space) "))
  (setq $ne-title (canonicalize (arg 2 ": new-entry (title) ") 1))
  (setq $ne-file (arg 3 ": new-entry (file) "))

```

```

(with-top-master-index
  (if (search-master-index $ne-space $ne-title)
      (message "Entry " $ne-space " " $ne-title " is already defined in "
                (current-buffer-name))
      (save-restriction
        (narrow-to-sub-index $ne-space)
        (end-of-file)
        (insert-string "\n\" " $ne-title "\"")
        (while (< (current-column) 65)
          (insert-string "\t") ; tab
        )
        (set-mark)
        (insert-string $ne-file "\n\n")
        (exchange-dot-and-mark)
        (relativize-file-name)
        (setq $ne-dict (concat (current-buffer-name) ":" $ne-space))
        (define-entry $ne-dict $ne-title $ne-file "")
        (put-in-dict $ne-dict $ne-title &entries)
      )
  )
)
))

```

```

(defun (delete-entry $de-space $de-name $de-index $de-title $de-dict
  (setq $de-space (arg 1 ": delete-entry (space) "))
  (setq $de-name (canonicalize (arg 2 ": delete-entry (name) ")))
  (with-top-master-index
    (if (! (setq $de-index (search-master-index $de-space $de-name)))
        (message "Entry " $de-space " " $de-name " is not defined in "
                  (current-buffer-name))
        (save-restriction
          (setq $de-title (get-entry-title $de-index))
          (narrow-to-sub-index $de-space)
          (beginning-of-file)
          (if (error-occurred (search-forward "\n\" " $de-title "\""))
              (message "Entry " $de-space " " $de-name " was not found in "
                        (current-buffer-name))
              (progn
                (beginning-of-line)
                (set-mark)
                (end-of-line)
                (if (error-occurred
                    (re-search-forward "^\"[^\"]*\"[ \t]+"))
                    (end-of-file)
                    (beginning-of-line)
                )
                (erase-region)
                (setq $de-dict (concat (current-buffer-name) ":" $de-space))
                (remove-from-dict $de-dict $de-index)
              )
          )
        )
    )
  )
)
))

```

```

(defun (new-synonym $ns-space $ns-title $ns-synonym $ns-index $ns-dict
  (setq $ns-space (arg 1 ": new-synonym (space) "))

```



```

    (progn
      (end-of-file)
      (set-mark)
      (exchange-dot-and-mark)
      0
    )
  )
  (progn
    (if (> (dot) (mark)) (exchange-dot-and-mark))
    1
  )
) ; if cache-field-markers:
(use-abbrev-table (concat (current-buffer-name) ":fields"))
(if (setq &ff-index (field-index $field-name))
  (progn
    (goto-character (array field-bottoms &ff-index))
    (error-occurred (backward-character))
    (set-mark)
    (goto-character (array field-tops &ff-index))
    (setq &ff-index &ff-index)
    (setq &ff-i 9999)
  )
)
)
(if (| (! &ff-index) ; not cached?
    (= (dot) (mark)) ; maybe collapsed?
  )
  (if (!= 0 (+ (execute-mlisp-string
                (concat
                  "(" entry-find-field-hook
                  " $field-name $field-create)"))))
    (progn ; field existed or was created, so cache it and return 1
      (if (> (dot) (mark))
        (exchange-dot-and-mark)
      ) ; now (dot) is at top of field, (mark) is at bottom of field
      (if (! &ff-index) ; found collapsed field?
        (progn ; reuse index & save in field index abbrev
          (setq &ff-index (setq &fields (+ &fields 1)))
          (temp-use-buffer entry-field-buffer
            (define-local-abbrev $field-name (concat &ff-index))
          )
        )
      )
    )
  )
  (if (< (array-size field-names) &fields)
    (setq field-names (grow-array field-names))
  )
  (setq-array field-names &ff-index $field-name)
  (if (< (array-size field-tops) &fields)
    (setq field-tops (grow-array field-names))
  )
  (setq-array field-tops &ff-index (dot))
  (if (< (array-size field-bottoms) &fields)
    (setq field-bottoms (grow-array field-names))
  )
  (exchange-dot-and-mark)
  (if (error-occurred (forward-character))
    (insert-string "\n")
  )
  )
  (setq-array field-bottoms &ff-index (dot))
  (backward-character)

```

```

        (exchange-dot-and-mark)
        1
      )
    (progn
      (end-of-file)
      (set-mark)
      0 ; field doesn't exist, so return 0
    )
  )
  1 ; fields exists in cache, so return 1
)
)
)
))

```

; (narrow-to-field "name") narrows to a field, if it exists.
; If there are more arguments, they're executed inside of a save-restriction
; narrowed to that field, and the restrictions are restored after executing
; them. Returns 1 upon success, or 0 if the field does not exist.

```

(defun (narrow-to-field &ntf-name
  (setq &ntf-name (arg 1 ": narrow-to-field (name) "))
  (widen-region)
  (if (!= (find-field &ntf-name 0) 0)
    (if (> (nargs) 1)
      (save-restriction
        (narrow-region)
        (progn-args 2)
        1
      )
      (progn (narrow-region) 1)
    )
    0
  )
)
)
))

```

```

(defun (entry-field-to-string &fts-name
  (setq &fts-name (arg 1 ": entry-field-to-string (name) "))
  (save-restriction
    (save-excursion
      (widen-region)
      (beginning-of-file)
      (if (!= (find-field &fts-name 0) 0)
        (region-to-string)
        ""
      )
    )
  )
)
)
))

```

```

(defun (string-to-entry-field &stf-name &stf-string
  (setq &stf-name (arg 1 ": string-to-entry-field (name) "))
  (setq &stf-string (arg 2 ":_string: "))
  (save-restriction
    (save-excursion
      (widen-region)
      (beginning-of-file)
      (if (!= (find-field &stf-name 1) 0)

```

```

        (replace-region &stf-string)
        (message "Can't find field " &stf-name)
    )
)
))

```

```

(defun (replace-region $rr-new
  (setq $rr-new (arg 1 ": replace-region "))
  (erase-region)
  (insert-string "|")
  (backward-character)
  (insert-string $rr-new)
  (delete-next-character)
))

```

```

(defun (forget-fields $ff-names $ff-fields $ff-i
  (setq $ff-names field-names)
  (setq $ff-fields &fields)
  (temp-use-buffer entry-field-buffer
    (while (<= (++ $ff-i) $ff-fields)
      (define-local-abbrev (array $ff-names $ff-i) 0)
    )
  )
  (setq &fields 0)
))

```

```

(defun (old-edit-field &ef-name &ef-string &ef-buf &ef-trbuf
  (setq &ef-name (arg 1 ": edit-field (name) "))
  (find-field &ef-name 1)
  (widen-region)
  (setq &ef-string (entry-field-to-string &ef-name))
  (setq &ef-buf (current-buffer-name))
  (setq &ef-trbuf (concat &ef-name ":" (current-buffer-name)))
  (edit-storyboard-contents) ; kludge! XXX
  (save-restriction
    (save-excursion
      (edit-in-transient-window &ef-trbuf
        (progn (erase-buffer)
          (pop-up-message &ef-string)
          ; (storyboard-mode)
          (insert-string &ef-string)
          (beginning-of-file)
        )
        (progn (setq &ef-string (buffer-to-string &ef-trbuf))
          (pop-up-message "did it")
          (string-to-entry-field &ef-name &ef-string)
        )
        (progn
          (message "Edit of " &ef-name " aborted.")
          (pop-up-message "argh")
        )
      )
    )
  )
  ; (edit-storyboard-contents) ; kludge! XXX
))

```

```

(defun (edit-field &ef-name
  &ef-string &ef-buf &ef-trbuf &ef-class &ef-title
  (setq &ef-name (arg 1 ": edit-field (name) "))
  (find-field &ef-name 1)
  (widen-region)
  (setq &ef-string (entry-field-to-string &ef-name))
  (setq &ef-buf (current-buffer-name))
  (setq &ef-trbuf (concat &ef-name ":" (current-buffer-name)))
  (setq &ef-class entry-class)
  (setq &ef-title (get-entry-title entry-index))
  (edit-storyboard-contents) ; kludge! XXX
  (#start-editor-window &ef-trbuf
    (setq @editor-save-action "#update-entry-field")
    (setq @editor-class-name &ef-class)
    (setq @editor-object-name &ef-name)
    (setq @editor-object-description (concat
      &ef-name " of " &ef-title
    ))
    (setq @editor-description (concat
      "You are editing the "
      "%$@editor-class-name of \""
      "%$@editor-object-name .\n"
      "When you exit the edit via %w" @editor-exit-action
      " , the field will be replaced by the string\n"
      "that you leave in the buffer.\n"
    ))
    (setq needs-checkpointing 0)
    (erase-buffer)
    (insert-string &ef-string)
    (beginning-of-file)
    (make-buffer-unmodified)
  )
))

; #update-entry-field is installed as the @editor-save-action in
; edit-field. It sets the field named by @editor-object-name, in
; the buffer named by @editor-client-buffer, to the string contained
; in the current buffer.
;
(defun (#update-entry-field &uef-string &uef-name
  (mark-whole-buffer)
  (setq &uef-string (region-to-string))
  (setq &uef-name @editor-object-name)
  (temp-use-buffer @editor-client-buffer
    (string-to-entry-field &uef-name &uef-string)
    (message "Updating " &uef-name " field of \""
      (get-entry-title entry-index) "\".")
    (edit-storyboard-contents)
  )
  (make-buffer-unmodified)
))

; -----
; Article stuff:
;

(auto-major-mode "storyboard-mode" "*.st0")

```

```

(declare-global narrow-storyboards)
(setq-default narrow-storyboards 1)

(use-syntax-table "storyboard"
  ; make just about everything be part of words
  (modify-syntax-entry "w      -+!$%^&=_/:?*.|a-zA-Z0-9#")

  ; add all the various types of parens.
  (modify-syntax-entry "({  {")
  (modify-syntax-entry " (  (")
  (modify-syntax-entry "(>  <")
  (modify-syntax-entry ") {  }")
  (modify-syntax-entry ") (  )")
  (modify-syntax-entry ")<  >")

  (modify-syntax-entry "  \") ; else forward/backward paren fails
  (modify-syntax-entry "  '"  ; else forward/backward paren fails
  (modify-syntax-entry "\\  \\" ; prefix char ; emacs "-)
;   (modify-syntax-entry "\"  ~")
)

(defun (storyboard-mode
  (pass-prefix-argument (#enter-major-mode "storyboard-mode"
    (progn
      (setq mode-string "storyboard")

      ; set up our context-specific modes
      (#storyboard-keybind)
      (use-abbrev-table "storyboard")
      (use-syntax-table "storyboard")
      (set-paragraph-delimiters)
      (set-indent-hook "#text-indent-hook")

      (setq &use-prefix-string 0)
      (setq auto-align-close-paren 0)
      (setq close-paren-deletes-space 0)

      (setq entry-author-hook "author-storyboard")
      (setq entry-instantiation-hook "instantiate-storyboard")
      (setq entry-find-field-hook "find-storyboard-field")

      (error-occurred (#storyboard-mode-hook))
;      (narrow-storyboard)
      (set-icon-image "no_ties")

    )
  ))
))

(defun (#storyboard-keybind
  (local-bind-to-key "flash-close-paren"      ") " ">" "}")
  (local-bind-to-key "forward-paren"          "\e)")
  (local-bind-to-key "backward-paren"          "\e(")
  (local-bind-to-key "storyboard-point-select" "\e ")
  (local-bind-to-key "storyboard-point-edit"  "\e\r")
  (local-bind-to-key "forward-button"          "\en")
  (local-bind-to-key "backward-button"         "\ep")
;   (set-point-select-hook "ht-point-define")

```

```

    (set-point-select-hook "storyboard-mouse-point-select")
    (set-point-edit-hook "storyboard-mouse-point-edit")
  ))

(defun (storyboard-mouse-point-select
; (save-excursion
; (save-window-excursion
;   (error-occurred (#to-mouse-context))
;   (storyboard-point-select)
; )
; )
))

(defun (storyboard-mouse-point-edit
; (save-excursion
; (save-window-excursion
;   (error-occurred (#to-mouse-context))
;   (storyboard-point-edit)
; )
; )
))

(defun (storyboard-point-select &sps-dot
; (setq &sps-dot (+ (dot)))
; (setq &sps-dot (dot))
; (if (bit& (count-tildes-above) 1)
;   (if (looking-at "~")
;       (progn (search-reverse "~")
;              (prompt-tilde-button)
;              )
;       (select-tilde-button)
;   )
;   (if (looking-at "~")
;       (prompt-tilde-button)
;       (if (& (! (looking-at "\\b\\.\\w+"))
;           (error-occurred (re-search-reverse "\\b\\.\\w+"))
;           (hit-background)
;           (select-dot-button &sps-dot)
;       )
;   )
; )
; (update-all-frames)
; (sit-for 0)
; (novalue)
))

(declare-global $disposition)
(setq $disposition "edit")

(defun (storyboard-point-edit
; (storyboard-point-dispose "edit")
; )
)

(defun (storyboard-point-define
; (storyboard-point-dispose "define")
; )
)

(defun (storyboard-point-dispose $disposition

```

```

(setq $disposition (arg 1))
(if (looking-at "~")
    (if (bit& (count-tildes-above) 1)
        (backward-character)
        (forward-character)
    )
    (if (looking-at "\\b\\.\\w+[ \\t\\n]*")
        (region-around-match 0)
    )
)
)
)
(storyboard-point-select)
))

(defun (hit-background
  (if making-link
    (finish-link)
  ; (link-out)
    (message "Thump!")
  )
)
)

(defun (select-tilde-button $argument
  (search-reverse "~")
  (forward-character)
  (set-mark)
  (search-forward "~")
  (backward-character)
  (setq $argument (region-to-string))
; (message-for 0 "~" $argument "~")
  (edit-article $argument)
)
)

(defun (prompt-tilde-button
  (.link)
)
)

(defun (select-dot-button &sdb-selected-point &sdb-starting-point
  &sdb-cmd &sdb-actions &sdb-i &sdb-j
  (setq &sdb-selected-point (arg 1))
  (setq &sdb-starting-point (dot))
  (region-around-match 0)
  (setq &sdb-cmd (canonicalize (region-to-string)))
; (pop-up-message &sdb-cmd)
  (setq &sdb-i (command-index &sdb-cmd))
  (if (<= &sdb-selected-point (dot))
    (if &sdb-i
        (execute-extended-command &sdb-cmd)
        (message "Unknown command: " &sdb-cmd)
    )
    (progn
      (setq &sdb-j 0)
      (while (& (! (bobp)) ; exhausted?
        (< &sdb-j 9999)) ; try doing one of the argument commands
        (if &sdb-i ; if known command
          (progn
            (setq &sdb-actions (array command-actions &sdb-i))
            (setq &sdb-j 0)
            (while (<= (++ &sdb-j) (array-size &sdb-actions))

```



```

; Returns 2 if .button
(defun (next-button
  (if (looking-at "\\b\\.\\w\\|~")
      (forward-character)
  )
  (if (bit& (count-tildes-above) 1)
      (error-occurred (search-forward "~"))
  )
  (if (error-occurred (re-search-forward "~\\|\\b\\.\\w"))
      0
      (progn
        (backward-character 1)
        (if (looking-at "~")
            (progn (forward-character 1) 1)
            (progn (backward-character 2) 2)
        )
      )
  )
))

; Moves cursor to previous button.
; Returns 0 if no next.
; Returns 1 if ~button~
; Returns 2 if .button
(defun (previous-button
  (if (bit& (count-tildes-above) 1)
      (error-occurred (search-reverse "~"))
  )
  (if (error-occurred (re-search-reverse "~\\|\\b\\.\\w"))
      0
      (if (looking-at "~")
          (if (error-occurred (search-reverse "~"))
              0
              (progn (forward-character) 1)
          )
          2
      )
  )
))

(defun (forward-button
  (if (! (next-button))
      (progn (beginning-of-file)
        (if (! (looking-at "\\b\\.\\w\\|~"))
            (next-button)
        )
      )
  )
))

(defun (backward-button
  (if (! (previous-button))
      (progn (end-of-file)
        (previous-button)
      )
  )
))

```

```

))

; Parse over one expression and return it as a string.
; Upon success, the first optional argument is executed with the region
; around the expression (not including surrounding white space or <>{}~
; delimiters), and the function may refer to and change the value of the
; string variable $expression, which will be used as the return value of
; gobble-expr. If the success function returns non-zero, the dot will be
; moved over any last delimiter, but if the function return zero, the dot
; will not be moved, and it is assumed the function left the dot where it
; thought it wanted it. (Hmm, with a stack, this could even be used to
; implement .include's of other buffers, or subroutines, or maybe even a
; real macro interpreter!) Upon encountering a close paren or end of file,
; it executes the second optional argument. If it encounters a syntax error,
; it executes the third optional argument.
; (gobble-expr
;   (handle-success $expression) ; => 0
;   (handle-end-of-expression)
;   (handle-syntax-error)
; ) ; => "The blue whale"
(defun (gobble-expr $expression
  (setq $expression "")
  (span-string-forward " \t\n")
  (if (looking-at "[<{}]")
    (progn (region-around-match 0)
      (set-mark)
      (backward-character)
      (if (error-occurred (forward-paren))
        (if (> (nargs) 2)
          (arg 3) ; syntax error
          (message "Syntax error: No matching paren!"))
        )
      (progn (backward-character)
        (setq $expression (region-to-string))
        (if (> (nargs) 0)
          (if (arg 1) (forward-character))
          (forward-character))
        )
      )
    )
  )
  (if (looking-at "~")
    (progn
      (forward-character)
      (set-mark)
      (if (error-occurred (search-forward "~"))
        (if (> (nargs) 2)
          (arg 3) ; syntax error
          (message "Syntax error: No matching ~"))
        )
      (progn (backward-character)
        (setq $expression (region-to-string))
        (if (> (nargs) 0)
          (if (arg 1) (forward-character))
          (forward-character))
        )
      )
    )
  )
)

```

```

    )
    (if (| (eobp)
        (! (looking-at "\\w+"))
    )
    (if (> (nargs) 1)
        (arg 2) ; eof or >}~ or ???
    )
    (progn (region-around-match 0)
        (setq $expression (region-to-string))
        (if (> (nargs) 0)
            (if (& (arg 1)
                eat-blank?
                (looking-at "[ \\t\\n]"))
            )
            (forward-character)
        )
        (if (& eat-blank?
            (looking-at "[ \\t\\n]"))
        )
        (forward-character)
    )
    )
    )
    )
    )
    $expression
))

(defun (replace-expr $re-new $re-rv
    (setq $re-new (arg 1 ": replace-expr "))
    (gobble-expr
        (progn
            (replace-region $re-new)
            (setq $re-rv 1)
        )
    )
    $re-rv
))

(defun (narrow-storyboard
))

(defun (instantiate-storyboard
    (setq entry-class "storyboard")
    (setq entry-field "contents")
    ; instantiate-storyboard-{article,picture,target}
    (execute-extended-command
        (concat "instantiate-storyboard-" (get-entry-space entry-index))
    )
    1
))

(defun (instantiate-storyboard-article ; for .st0
))

```

```

(defun (instantiate-storyboard-picture ; for later .pnl
))

(defun (instantiate-storyboard-target ; for later .tnl
))

(defun (author-storyboard
  (error-occurred (set-main-menu "hyperties"))
))

(defun (edit-article &ea-name &ea-def
  (setq &ea-name (arg 1 ": edit-article (name) "))
  (if (= $disposition "define")
    (progn
      (save-excursion
        (save-window-excursion
          (with-entry "article" &ea-name
            (setq &ea-def
              (concat
                "Definition of \""
                (get-entry-title entry-index)
                "\":\n"
                (entry-field-to-string "definition")
              )
            )
          )
        )
      )
    )
    (typeout-text &ea-def)
  )
  ; (frame-to-top (current-frame)) ; xxx
  )
  (progn
    (pop-to-buffer
      (with-entry "article" &ea-name
        (author-entry)
        (current-buffer-name)
      )
    )
    (save-command-context)
  )
)
))

(defun (edit-storyboard-definition
  (edit-field "definition")
))

; Add brains to this
(defun (edit-storyboard-synonyms
  (edit-field "synonyms")
))

; Add brains to this
(defun (edit-storyboard-notes
  (edit-field "notes")
))

; Reset and reinitialize a bit

```

```

(defun (edit-storyboard-contents
  (forget-fields)
  (author-entry)
))

(defun (make-storyboard $ms-file $ms-title $ms-index $ms-buf
  (setq $ms-file (arg 1 ": make-storyboard (filename) "))
  (if (! (index ".st" $ms-file))
    (setq $ms-file (concat $ms-file ".st0"))
  )
  (setq $ms-title (arg 2 ": make-storyboard (title) "))
  (new-entry "article" $ms-title $ms-file)
  (setq $ms-index (search-master-indices $ms-title "article"))
  (setq $ms-buf (get-entry-buffer $ms-index))
  (temp-use-buffer $ms-buf
    (erase-buffer)
    (insert-string
".title <" $ms-title ">\n\n"
".synonyms <\n>\n\n"
".definition <\n>\n\n"
".contents <\n>\n\n"
    )
    (edit-storyboard-contents)
  )
))

(defun (make-picture $mp-file $mp-title $mp-index $mp-buf
  (setq $mp-file (arg 1 ": make-picture (filename) "))
  (if (! (index ".pn" $mp-file))
    (setq $mp-file (concat $mp-file ".pn0"))
  )
  (setq $mp-title (arg 2 ": make-picture (title) "))
  (new-entry "picture" $mp-title $mp-file)
  (setq $mp-index (search-master-indices $mp-title "picture"))
  (setq $mp-buf (get-entry-buffer $mp-index))
  (temp-use-buffer $mp-buf
    (erase-buffer)
    (author-entry)
  )
))

(defun (make-target $mt-file $mt-title $mt-index $mt-buf
  (setq $mt-file (arg 1 ": make-target (filename) "))
  (if (! (index ".tn" $mt-file))
    (setq $mt-file (concat $mt-file ".tn0"))
  )
  (setq $mt-title (arg 2 ": make-target (title) "))
  (new-entry "target" $mt-title $mt-file)
  (setq $mt-index (search-master-indices $mt-title "target"))
  (setq $mt-buf (get-entry-buffer $mt-index))
  (temp-use-buffer $mt-buf
    (erase-buffer)
    (author-entry)
  )
))

; (find-storyboard-field $field-name $field-create)
(defun (find-storyboard-field &fsf-name &fsf-create

```

```

    &fsf-target &fsf-expr &fsf-ret
    (setq &fsf-name (arg 1 ":_field name: "))
    (setq &fsf-create (arg 2 ":_create: "))
    (setq &fsf-target (concat "\\b\\." &fsf-name "\\b"))
    (setq &fsf-expr "")
; (message-for 10 "FSF BEGIN")
    (beginning-of-file)
    (if (error-occurred (re-search-forward &fsf-target))
        (if &fsf-create
            (progn
                (widen-region)
                (end-of-file)
                (insert-string "\n." &fsf-name "\n<>\n")
                (backward-character)
                (backward-character)
                (set-mark)
                1
            )
            0
        )
        (progn
            (region-around-match 0)
            (setq &fsf-ret 0)
            (gobble-expr (progn (setq &fsf-ret 1) 0))
            &fsf-ret
        )
    )
))

(defun (count-tildes-above &tildes
    (save-excursion
        (setq &tildes 0)
        (while (! (error-occurred (search-reverse "~")))
            (++ &tildes)
            (error-occurred
                (backward-character)
                (if (looking-at "\\") ; emacs ") - :
                    ( -- &tildes)
                    (forward-character)
                )
            )
        )
        &tildes
    )
))

; -----
; Command callbacks:
;

(defun (.link $article
    (save-excursion (setq $article (gobble-expr)))
    (get-default-space-word "Enter .link " $article "article")
))

(defun (.picture $picture
    (setq $picture (gobble-expr))
    (get-default-space-word "Enter .picture " $picture "picture")

```

```

))

(defun (.target $target $article
  (setq $target (gobble-expr))
  (setq $article (gobble-expr))
  (get-default-space-word "Enter .target " $target "target")
  (get-default-space-word "Enter .target <" $target "> link "
    $article "article")
))

(defun (.file $filename
  (setq $filename (gobble-expr))
  (edit-file $filename)
))

(defun (.directory $filename
  (setq $filename (gobble-expr))
  (edit-file $filename)
))

(defun (edit-file $ef-name
  (setq $ef-name (arg 1 ":_name: "))
  (if (!= (substr $ef-name 1 1) "/")
    (setq $ef-name (concat (get-entry-directory entry-index)
      "/" $ef-name))
  )
  (find-file $ef-name)
))

(defun (.fork
  (message "Enter .fork <" (gobble-expr) ">")
))

(defun (fork-program $arg
  (setq $arg (arg 1 ": fork-program "))
  (run-program $arg)
))

(defun (.psh
  (message "Enter .psh <" (gobble-expr) ">")
))

(defun (psh-file $arg
  (setq $arg (arg 1 ": psh-file "))
  (run-program (concat "psh " $arg))
))

(defun (.psview
  (message "Enter .psview <" (gobble-expr) ">")
))

(defun (psview-file $arg
  (setq $arg (arg 1 ": psview-file "))
  (run-program (concat "psview " $arg))
))

(defun (.paper
  (message "Enter .paper <" (gobble-expr) ">")
)

```

```

))

(defun (paper-file $arg
  (setq $arg (arg 1 ": paper-file "))
  (run-program (concat "paper " $arg))
))

(defun (run-program &rp-dir &rp-cmd
  (setq &rp-dir (file-directory-name (current-file-name)))
  (setq &rp-cmd
    (concat "cd " &rp-dir " ; "
      (arg 1 ": run-program ")))
  (execute-shell-command &rp-cmd "*cmd*")
))

(defun (.postscript
  (message "Enter .postscript <" (gobble-expr) ">")
))

(defun (postscript-button $arg
  (setq $arg (arg 1 ": postscript-button "))
  (execute-ps-string $arg)
  (message "Executed postscript code " $arg)
))

(defun (.mlisp
  (message "Enter .mlisp <" (gobble-expr) ">")
))

(defun (mlisp-button $arg
  (setq $arg (arg 1 ": mlisp-button "))
  (if (error-occurred (execute-mlisp-string $arg))
    (message "Error executing mlisp code " $arg)
    (message "Executed mlisp code " $arg)
  )
))

(defun (.net
  (message "Enter .net <" (gobble-expr) ">")
))

(defun (net-button $arg
  (setq $arg (arg 1 ": net-button "))
  (message "TODO: .net " $arg)
))

(defun (.button
  (message
    "Enter .button <" (gobble-expr) "> <" (gobble-expr) "> <" (gobble-expr) ">"
  )
))

(defun (edit-label $old $new
  (save-excursion
    (setq $old (arg 1))
    (push-back-string $old)
    (setq $new (get-tty-string ": edit-label (new label) "))
  )
)

```



```

    (erase-region)
    (insert-string $new)
  ))

(defun (popup-string $arg
  (setq $arg (arg 1 ": popup-string "))
  (pop-up-message $arg)
))

(defun (typeout-string $arg
  (setq $arg (arg 1 ": typeout-string "))
  (typeout-text $arg)
))

(defun (edit-string $arg $buf
  (setq $arg (arg 1 ": edit-string "))
  (setq $buf (current-buffer-name))
  (edit-in-transient-window
    (concat "string:" (current-buffer-name))
    (progn (erase-buffer)
      (insert-string $arg)
      (beginning-of-file)
    )
    (progn (setq $arg (buffer-to-string (current-buffer-name)))
      (erase-buffer)
      (switch-to-buffer $buf)
      (erase-region)
      (insert-string $arg)
    )
    )
  (message "Aborted edit of string")
)
))

```

```

; making-link state:
; 0 : not making link
; 1 : making in link
; 2 : making out link
; 3 : making inout link
(declare-global making-link)
(declare-global @link-start)
(declare-global $link-start)

```

```

(defun (link-in
  (start-link 1)
)
)

```

```

(defun (link-out
  (start-link 2)
)
)

```

```

(defun (link-inout
  (start-link 3)
)
)

```

```

(defun (make-link
  (link-out)
)
)

```

```

))

(defun (start-link
  (setq making-link (arg 1))
  (setq @link-start (dot))
  (setq $link-start (get-entry-title entry-index))
  (if (bit& making-link 2)
      (progn
        (set-mark)
        (insert-string "~!article~ ")
        (exchange-dot-and-mark)
      )
    )
  (message "Starting link from " $link-start)
  (set-mouse-button-up-hook "start-link-up")
))

(defun (start-link-up
  (if (= $link-start (get-entry-title entry-index))
      (progn
        (message "bonk!")
      )
    (progn
      (finish-link)
    )
  )
))

(defun (finish-link $link-finish
  (setq $link-finish (get-entry-title entry-index))
  (if (bit& making-link 1)
      (progn
        (if (bit& (count-tildes-above) 1)
            (search-reverse "~")
          )
        (insert-string "~" $link-start "~ ")
      )
    )
  (if (bit& making-link 2)
      (temp-use-buffer @link-start
        (goto-character @link-start)
        (gobble-expr
          (progn
            (erase-region)
            (insert-string "~" $link-finish "~ ")
            (switch-to-frame entry-frame)
            (save-command-context)
            (frame-to-top entry-frame)
          )
        )
      )
    )
  (message
    "Made link: " $link-start
    (if (= making-link 1)
        " <= "
    )
  )
)

```

```

        (= making-link 2)
        " => "
        (= making-link 3)
        " <=> "
    ) $link-finish
)
(setq making-link 0)
))

(defun (select-link $link-from $link-to $link-name

))

; -----
; Picture stuff:
;

(auto-major-mode "picture-author-mode" "*.pn0")

(defun (picture-author-mode
  (pass-prefix-argument (#enter-major-mode "picture-author-mode"
    (progn
      (setq mode-string "picture-author")

      ; set up our context-specific modes
      ; (#storyboard-keybind)
      ; (use-abbrev-table "storyboard")
      ; (use-syntax-table "storyboard")
      (set-paragraph-delimiters)
      (set-indent-hook "#text-indent-hook")

      (setq &use-prefix-string 0)
      (setq auto-align-close-paren 0)
      (setq close-paren-deletes-space 0)

      (setq entry-author-hook "author-picture")
      (setq entry-instantiation-hook "instantiate-picture")
      (setq entry-find-field-hook "find-picture-field")

      (error-occurred (#picture-author-mode-hook))
      (set-icon-image "no_ties")

    )
  ))
))

(defun (instantiate-picture
  (setq entry-field "") ; whole buffer
  1
)

)

(defun (author-picture
  (error-occurred (set-main-menu "hyperties-picture"))
  (save-excursion
    (beginning-of-file)
    (set-mark)
    (end-of-line)
    (setq entry-class (region-to-string))

```

```

    )
  ))

(defun (edit-picture &ep-name
  (setq &ep-name (arg 1 ": edit-picture (name) "))
  (pop-to-buffer
    (with-entry "picture" &ep-name
      (author-entry)
      (current-buffer-name)
    )
  )
  (save-command-context)
))

(defun (find-picture-field
  (mark-whole-buffer)
  1
))

(defun (ht-view-picture
  (message "TODO: ht-view-picture")
))

(defun (ht-edit-picture-canvas $scan-name $scan-relname
  (if (| (= entry-class "Raster")
    (= entry-class "ScaledRaster")
  )
  (progn
    (get-can-name)
    (run-program
      (concat
        picture-editor " "
        $scan-relname " "
        (get-entry-directory entry-index)
        " "
        (if (= entry-class "ScaledRaster")
          (save-excursion
            (beginning-of-file)
            (re-search-forward "[0-9]+ [0-9]+$")
            (region-around-match 0)
            (region-to-string)
          )
          ""
        )
      )
    )
  )
  (| (= entry-class "Picture")
    (= entry-class "ScaledPicture")
  )
  (progn
    (message "TODO: Picture editor")
  )
)
))

(defun (ht-make-picture-Picture
  (message "TODO: ht-make-picture-Picture")

```

```

))

(defun (ht-make-picture-UnitPicture
  (message "TODO: ht-make-picture-UnitPicture")
))

(defun (get-can-name
  (setq $scan-name (get-entry-filename entry-index))
  (temp-use-buffer "*scratch*"
    (erase-buffer)
    (insert-string $scan-name)
    (goto-character (- (dot) 4))
    (if (looking-at "\\\.pn.")
      (kill-to-end-of-line)
    )
    (insert-string ".can")
    (setq $scan-name (buffer-to-string "*scratch*"))
    (search-reverse "/" )
    (forward-character)
    (set-mark)
    (end-of-file)
    (setq $scan-relname (region-to-string))
  )
))

(defun (get-can-from-screen
  (execute-ps-string
    (concat
      "{ framebuffer createoverlay setcanvas getwholerect waitprocess aload pop "
      " framebuffer setcanvas points2rect rectpath (\" $scan-name \") writescreen "
      "} fork pop"
    )
  )
  (message-for 1 "Drag the rectangle over the part of the screen you want.")
  (update-current-frame)
))

(defun (ht-make-picture-Raster $scan-name $scan-relname
  (get-can-name)
  (erase-buffer)
  (insert-string
    "Raster\n"
    "\"\" (get-entry-title entry-index) \"\"\n"
    "(" $scan-relname ") here findraster\n"
  )
  (author-picture)
  (get-can-from-screen)
))

(defun (ht-make-picture-ScaledRaster $scan-name $scan-relname $scan-w $scan-h
  (setq $scan-w (get-tty-string "Enter canvas width: "))
  (setq $scan-h (get-tty-string "Enter canvas height: "))
  (get-can-name)
  (erase-buffer)
  (insert-string
    "ScaledRaster\n"
    "\"\" (get-entry-title entry-index) \"\"\n"
    "(" $scan-relname ") here findraster\n"
  )

```

```

    $scan-w " " $scan-h "\n"
  )
  (author-picture)
  (get-can-from-screen)
))

; -----
; Target stuff:
;

(declare-buffer-specific "target-picture-name")
(setq-default target-picture-name 0)

(auto-major-mode "target-author-mode" "/*.tn0")

(defun (target-author-mode
  (pass-prefix-argument (#enter-major-mode "target-author-mode"
    (progn
      (setq mode-string "target-author")

      ; set up our context-specific modes
      (#target-author-keybind)
      (use-abbrev-table "storyboard")
      (use-syntax-table "storyboard")
      (set-paragraph-delimiters)
      (set-indent-hook "#text-indent-hook")

      (setq &use-prefix-string 0)
      (setq auto-align-close-paren 0)
      (setq close-paren-deletes-space 0)

      (setq entry-author-hook "author-target")
      (setq entry-instantiation-hook "instantiate-target")
      (setq entry-find-field-hook "find-target-field")

      (error-occurred (#target-author-mode-hook))
      (set-icon-image "no_ties")

    )
  ))
))

(defun (#target-author-keybind
  (local-bind-to-key "self-insert" "\n")
))

(defun (instantiate-target
  (setq entry-field "") ; whole buffer
  1
))

(defun (author-target
  (error-occurred (set-main-menu "hyperties-target"))
  (save-excursion
    (beginning-of-file)
    (set-mark)
    (end-of-line)
    (setq entry-class (region-to-string))

```

```

))
(defun (edit-target &set-name
  (setq &set-name (arg 1 ": edit-target (name) "))
  (pop-to-buffer
    (with-entry "target" &set-name
      (author-entry)
      (current-buffer-name)
    )
  )
  (save-command-context)
))

(defun (find-target-field
  (mark-whole-buffer)
  1
))

(defun (make-target-Target $target-picture $scan-name $scan-relname $scan-dir
  (setq $target-picture (arg 1 ": make target (picture name) "))
  (if (! (search-master-indices $target-picture "target"))
    (message "Unknown picture " $target-picture)
    (progn
      (with-entry "picture" $target-picture
        (get-can-name)
        (setq $scan-dir (get-entry-directory entry-index))
      )
      (run-program
        (concat
          target-editor " "
          $scan-relname " "
          $scan-dir " "
          (file-leaf-name (get-entry-filename entry-index)) " "
          (file-directory-name (get-entry-filename entry-index)) " "
          (get-entry-title entry-index) " "
          entry-frame
        )
      )
    )
  )
)

(defun (make-target-Link
  (message "TODO: make-target-Link")
)

(defun (make-target-Menu
  (message "TODO: make-target-Menu")
)

(defun (make-target-PageTracker
  (message "TODO: make-target-PageTracker")
)

(defun (make-target-Popup
  (message "TODO: make-target-Popup")
)

```

```

(defun (make-target-NewPopup
  (message "TODO: make-target-NewPopup")
))

(defun (make-target-Scrollbar
  (message "TODO: make-target-Scrollbar")
))

(defun (make-target-Slider
  (message "TODO: make-target-Slider")
))

(defun (make-target-TextEdit
  (message "TODO: make-target-TextEdit")
))

(defun (make-target-TextCanvas
  (message "TODO: make-target-TextCanvas")
))

(defun (make-target-Animated
  (message "TODO: make-target-Animated")
))

(defun (make-target-Emacs
  (message "TODO: make-target-Emacs")
))

(defun (#enter-target-body $etb-name
  (setq $etb-name (arg 1 ": #enter-target-body (name) "))
  (pop-up-message $etb-name)
  (if (search-master-indices $etb-name "target")
    (with-entry "target" $etb-name
      (switch-to-frame entry-frame)
      (save-command-context)
      (erase-buffer)
      (recursive-edit)
      (author-target)
      (message "Entered target body " $etb-name)
    )
    (temp-use-buffer "bogus-target-body"
      (switch-to-frame 0)
      (save-command-context)
      (erase-buffer)
      (recursive-edit)
      (message "Entered bogusly named target into bogus-target-body")
    )
  )
))

; -----
; Command interpreter interface:
;
; Commands are defined in a file with the following syntax, which
; specified the number of arguments of each command, the name space
; of each argument (used to give help and completion when prompting),
; and the name of the function to call when the argument is selected.

```



```

;
; .link <article:edit-article>
; .picture <picture:edit-picture>
; .target <target:edit-target> <article:edit-article>
; .file <file:edit-file>
; .fork <string:fork-program>

(declare-global hyperties-command-buffer)

(defun (init-commands
  &ic-cmd &ic-expr &ic-space &ic-action &ic-array &ic-args
  &ic-i &ic-split &ic-spaces
  (setq hyperties-command-buffer (current-buffer-name))
  (use-syntax-table "storyboard")
  (setq &ic-array nullarray)
  (setq &ic-cmd "")
  (beginning-of-file)
  (setq &commands 0)
  (use-abbrev-table hyperties-command-buffer)
  (while (≠ (setq &ic-cmd (gobble-expr)) "")
    (if (= &ic-cmd "#")
      (progn ; skip rest of line if it's a comment
        (next-line)
        (if (≠ (eobp))
          (beginning-of-line)
        )
      )
    )
    (progn ; read and parse command arguments
      (++ &commands)
      (if (< (array-size command-names) &commands)
        (setq command-names (grow-array command-names))
      )
      (define-local-abbrev
        (setq-array command-names &commands (canonicalize &ic-cmd))
        &commands
      )
      (if (< (array-size command-spaces) &commands)
        (setq command-spaces (grow-array command-spaces))
      )
      (setq-array command-spaces &commands nullarray)
      (if (< (array-size command-actions) &commands)
        (setq command-actions (grow-array command-actions))
      )
      (setq-array command-actions &commands nullarray)
      (setq &ic-args 0)
      (if (= &ic-array nullarray)
        (setq &ic-array (new-array #args))
      )
      (while (& (≠ (setq &ic-expr (gobble-expr)) ""))
        (if (= (substr &ic-expr 1 1) ".")
          (progn (exchange-dot-and-mark) ; move before .cmd
            0 ; stop loop
          )
          1 ; continue loop
        )
      )
      (if (<= (array-size &ic-array) &ic-args)
        (setq &ic-array (grow-array &ic-array))
      )
    )
  )

```



```

        (beginning-of-file)
        (if &c-fold (case-region-lower))
        (error-occurred
         (re-replace-string "[ \n\t<>~]+" " "))
        (beginning-of-file)
        (if (looking-at " ") (delete-next-character))
        (end-of-file)
        (error-occurred
         (backward-character)
         (if (looking-at " ") (delete-next-character)))
        (buffer-to-string "*canonicalize*")
    )
))

; Stop writing those stupid *canonicalize*.CKP files!
(temp-use-buffer "*canonicalize*"
 (setq needs-checkpointing 0)
)

(declare-global old-arrays)
(if (error-occurred (array-size old-arrays))
    (setq old-arrays nullarray)
)

(defun (free-array $fa-array
  (setq $fa-array (arg 1))
  (if (! (error-occurred (array $fa-array 1)))
    (progn
      (setq-array $fa-array 1 old-arrays)
      (setq old-arrays $fa-array)
    )
  )
)
))

; Deactivate the above 'cause it's broke!
(defun (free-array
)

; Try to find an best fit old array at least as big as a given size. If no
; suitable free array is found, make a new one.
; (old-array 100) => array[180]
(defun (old-array $oa-size $oa-array $oa-parent $oa-result $oa-resultparent
  (setq $oa-size (arg 1 ":_size: "))
  (setq $oa-array old-arrays)
  (setq $oa-parent nullarray)
  (setq $oa-result nullarray)
  (setq $oa-resultparent nullarray)
  (while (!= $oa-array nullarray) ; is it the end of the list?
    (if (>= (array-size $oa-array) $oa-size) ; big enough?
      (progn ; this array we can use
        (if (| (= $oa-result nullarray)
              (< (array-size $oa-array)
                 (array-size $oa-result)))
          )
        (progn ; this array we'd like to use
          (setq $oa-resultparent $oa-parent)
          (setq $oa-result $oa-array)
        )
      )
    )
  )
)

```

```

    )
  )
)
(progn ; try next array
  (setq $oa-parent $oa-array)
  (setq $oa-array (array $oa-array 1))
)
)
(if (= $oa-result nullarray) ; was no suitable free array found?
  (new-array $oa-size) ; make brand new array
  (progn
    (if (= $oa-resultparent nullarray) ; 1st array on free list?
      (setq old-arrays (array old-arrays 1)) ; replace free list
      (setq-array $oa-resultparent 1
        (array $oa-result 1)) ; replace next
    )
    (setq-array $oa-result 1 0) ; zorch reference to rest of free list
    $oa-result ; this array we'll use
  )
)
)
))

```

```

(defun (ensure-array $ea-array $ea-size
  (setq $ea-array (arg 1))
  (setq $ea-size (arg 2))
  (if (| (error-occurred (array $ea-array 1))
    (< (array-size $ea-array) $ea-size)
  )
    (progn (free-array $ea-array)
      (old-array $ea-size)
    )
    $ea-array
  )
)
)

```

```

; Return a copy of an array, but bigger.
; (grow-array old-array) => new-array
(defun (grow-array $ga-array $ga-new-array $ga-size $ga-i
  (setq $ga-array (arg 1))
  (setq $ga-size (array-size $ga-array))
  (setq $ga-new-array (old-array (+ $ga-size $ga-size 2)))
  (setq $ga-i 0)
; (message-for 1 "Growing array from " $ga-size
; " to " (array-size $ga-new-array))
  (while (<= (++ $ga-i) $ga-size)
    (setq-array $ga-new-array $ga-i (array $ga-array $ga-i)))
  (free-array $ga-array)
; (message-for 1 "Grew array from " $ga-size
; " to " (array-size $ga-new-array))
  $ga-new-array
)
)

```

```

; (with-entry "article" "!home" ...)
(defun (with-entry &we-space &we-name &we-index
  (save-excursion
    (save-window-excursion
      (setq &we-space (arg 1))
      (setq &we-name (arg 2))
    )
  )
)

```

```

    (setq &we-index (search-master-indices &we-name &we-space))
    (if &we-index
        (progn (switch-to-buffer (get-entry-buffer &we-index))
                (progn-args 2)
                )
        (message "Can't find it!")
    )
  )
)
))

(defun (httest
  (save-excursion
    (save-window-excursion
      (init-hyperties)
      (visit-file "/tumtum/don/db/commands")
      (init-commands)
      (if (buffer-exists "master-index")
          (delete-buffer "master-index"))
      (if (buffer-exists "master-index<2>")
          (delete-buffer "master-index<2>"))
      (if (buffer-exists "master-index<3>")
          (delete-buffer "master-index<3>"))
      (if (buffer-exists "master-index<4>")
          (delete-buffer "master-index<4>"))
      (if (buffer-exists "master-index<5>")
          (delete-buffer "master-index<5>"))
      ; (push-master-index-file "/tumtum/guest/cties/objects/master-index")
      (push-master-index-file "/tumtum/don/db/master-index")
      (edit-article "!home")
    )
  )
))

(defun (author
  (save-excursion
    (save-window-excursion
      (init-hyperties)
      (visit-file "/tumtum/don/db/commands")
      (init-commands)
      (if (buffer-exists "master-index")
          (delete-buffer "master-index"))
      (if (buffer-exists "master-index<2>")
          (delete-buffer "master-index<2>"))
      (if (buffer-exists "master-index<3>")
          (delete-buffer "master-index<3>"))
      (if (buffer-exists "master-index<4>")
          (delete-buffer "master-index<4>"))
      (if (buffer-exists "master-index<5>")
          (delete-buffer "master-index<5>"))
      (push-master-index-file "/tumtum/guest/cties/objects/master-index")
      (push-master-index-file "/tumtum/guest/cties/databases/author/master-index")
      (edit-article "!home")
    )
  )
))

(defun (funtest

```

```

(save-excursion
  (save-window-excursion
    (init-hyperties)
    (visit-file "/tumtum/don/db/commands")
    (init-commands)
    (if (buffer-exists "master-index")
        (delete-buffer "master-index"))
    (if (buffer-exists "master-index<2>")
        (delete-buffer "master-index<2>"))
    (if (buffer-exists "master-index<3>")
        (delete-buffer "master-index<3>"))
    (if (buffer-exists "master-index<4>")
        (delete-buffer "master-index<4>"))
    (if (buffer-exists "master-index<5>")
        (delete-buffer "master-index<5>"))
    (push-master-index-file
      "/tumtum/guest/cties/databases/fun/master-index")
    (edit-article "!home")
  )
)
))

; Zap this once buffer-file-name autoloads buffer.ml:
(buffer-is-scratch "This is here to get buffer.ml autoloaded...")

```