

```

; ties.ml      Emacs support for HyperTies
;

;-----
; interactive ties session (separate from the terminal's server connection)
;-----

(setq-option-default ties-directory      "~don/n/ties/")
(if (| (= (system-name) "dot")
      (= (system-name) "pink"))
    (progn
      (setq-option-default ties-program (concat
                                         "rsh repo 'cd " ties-directory
                                         " ; setenv NEWSSERVER \"
                                         (getenv "NEWSSERVER")
                                         "\" ; f ")")
      (setq-option-default ties-arguments (concat " ties/global/browse -'"))
      (progn
        (setq-option-default ties-program (concat
                                           ties-directory
                                           "f "))
        (setq-option-default ties-arguments (concat " ties/global/browse -")))))
(setq-option-default ties-init-string ".f\n")
(setq-option-default ties-buffer      "ties")
(setq-option-default ties-icon        "bulb_1")

; ties starts a ties window, or pops to it if it is already started.  If not it
; starts the program named by ties-program.  Either way, it pops up an
; interactive process window on the ties listener.  Then it sets up the ties
; engine output dispatcher and sentinel so Emacs facilities can receive
; messages from the ties session and detect when the engine has terminated.
;
(defun (ties $db $started
  (setq $db (arg 1 ": ties (database) "))
  (start-interactive-process
    (concat "exec " ties-program $db ties-arguments)
    ties-buffer
    (progn
      ; the following is done only if the process was not already
      ; running
      (postscript-mode)
      (listener-mode)
      (#process-do-key-bindings)
      (setq $started 1)
      (setq wrap-long-lines 0)
    )
  )
  ; the init string is now sent by the #prescript-entering function
  (if $started
    (progn
      (string-to-process (active-process)
        ties-init-string
      )
      (message "Started " ties-program)
      (set-primary-icon-image ties-icon)
    )
  )
))

```

```

(defun (execute-ties-buffer
  (string-to-process "ties" (buffer-to-string (current-buffer-name)))
))
(defun (execute-ties-string
  (string-to-process "ties"
    (if (interactive)
      (arg 1 ": execute-ties-string ")
      (pass-mlisp-args 1 -1 (concat))
    )
  )
)
))
(defun (execute-ties-def
  (save-excursion
    (mark-postscript-def)
    (narrow-bounds-region
      (execute-ties-buffer)
    )
  )
)
))

(defun (ties-format-node $tfn-title
  (setq $tfn-title &info-current-node)
  (copy-current-buffer "*ties-scratch*")
  (temp-use-buffer "*ties-scratch*"
    (save-excursion
      (beginning-of-file)
      (set-mark)
      (search-forward "\n\n")
      (narrow-bounds-region
        (beginning-of-file)
        (re-replace-string "\\(: *\\)\\([^\t\n]+\\)"
          "\\1 .~ >\\2~"))
      (beginning-of-file)
      (error-occurred
        (re-replace-string "\n*\\* *[Mm]enu\\:\\n*" "\n.nl\n")
        (re-replace-string "^\\* *\\([^\t\n]*\\)"
          ".nl .~ -\\1~"))
      (error-occurred
        (replace-string "\n\n" "\n.nl\n.nl\n"))
      (error-occurred
        (re-replace-string "\\* *[Nn]ote *\\([^\n]*\\):"
          " .~ *\\1~:"))
      (insert-string
        (concat ".start-article\n.title\n"
          $tfn-title
          "\n.contents\n"))
      (end-of-file)
      (insert-string "\n.end\n.end-article\n")
      (write-named-file "~/ .ties-scratch")
      (string-to-process "ties"
        (concat "fload " (expand-file-name "~/ .ties-scratch") "\n"))
      )
    )
  )
)

(defun (ties-format-definition
  (string-to-process "ties"
    (concat ".start-definition\n.title\n"

```

```

        (arg 1 ": ties-format-definition (title) ")
        "\n.definition\n"
        (arg 2 ": ties-format-definition (text) ")
        "\n.end\n.end-definition\n"))))

(defun (ties-format-article
  (string-to-process "ties"
    (concat ".start-article\n.title\n"
      (arg 1 ": ties-format-article (title) ")
      "\n.contents\n"
      (arg 2 ": ties-format-article (text) ")
      "\n.end\n.end-article\n"))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; magic string format
; &      keystrokes read as Emacs commands in current frame context
; :      MLisp expression evaluated in current frame context
; !      shell command
; ^      input to ties formatter
; *      info-go-menu-item
; >      info-go-node
; else    message
;

(defun (ties-info-go-menu-item $s
  (setq $s (arg 1 ": ties-info-go-menu-item "))
  (info-go-menu-item $s)
  (ties-format-node)
))

(defun (ties-info-go-footnote $s
  (setq $s (arg 1 ": ties-info-go-footnote "))
  (info-go-footnote $s)
  (ties-format-node)
))

(defun (ties-info-go-node $tign-s
  (setq $tign-s (arg 1 ": ties-info-go-node "))
  (info-go-node $tign-s)
  (ties-format-node)
))

(defun (ties-dot-command $tidc-s
  (setq $tidc-s (arg 1 ": ties-dot-command ."))
  (string-to-process "ties"
    (concat "." $tidc-s "\n")
  )
))

(defun (ties-command $tc-s
  (setq $tc-s (arg 1 ": ties-command "))
  (string-to-process "ties"
    (concat $tc-s "\n")
  )
))

```

```

; Need to also send .pile <n>
(defun (ties-define $td-s
  (setq $td-s (arg 1 ": ties-define "))
  (string-to-process "ties"
    (concat ".define " $td-s "\n")
  )
))

; Need to also send .pile <n>
(defun (ties-articulate $ta-s
  (setq $ta-s (arg 1 ": ties-articulate "))
  (string-to-process "ties"
    (concat ".articulate " $ta-s "\n")
  )
))

; Need to also send .pile <n>
(defun (ties-return
  (string-to-process "ties" ".return\n")
))

(defun (ties-shell-command
  (execute-shell-command (arg 1 ": ties-shell-comand ")
    "ties-shell-command"
  )
))

(defun (ties-message
  (string-to-process "ties"
    (concat (char-to-string (last-key-struck)))
    (arg 1 ": ties-message ")
  )
))

(defun (make-ties-window $mtw-frame
  (set-default-frame-class "TiemacsWindow")
  (save-excursion
    (setq $mtw-frame (create-frame))
    (switch-to-frame $mtw-frame)
    (switch-to-buffer (concat "*ties-" $mtw-frame "*"))
    (update-frame))
  (execute-ties-string (concat
    ".pile-pos " (arg 1 "x: ") " " (arg 2 "y: ") "\n"
    ".pile-size " (arg 3 "w: ") " " (arg 4 "h: ") "\n"
    ".new-pile " $mtw-frame "\n"
  ))
  (set-default-frame-class "TabEmacsWindow")
))

(defun (tiemacs
  (setq ties-program (concat ties-directory "fe "))
  (ties)
))

(defun (make-tiemacs-windows
  (execute-ties-string ".init\n")
  (make-ties-window 575 220 576 680)
  (execute-ties-string ".name-pile RightBrowser\n")
  (execute-ties-string ".name-pile LeftBrowser\n")
)

```



```
; Authoring modes

/* Window flag bits, stored in winflags
; */
#define      WF_frozen          1      /* resist resize and takeover */
#define      WF_guarded        2      /* resist trespass */
#define      WF_inverse        4      /* inverse-video entire buffer */
#define      WF_nomodeline     8      /* don't display modeline */
#define      WF_topmodeline    16     /* show modeline on top */
#define      WF_noinvmodeline  32     /* don't inverse-video modeline */

(auto-execute "ties-storyboard-mode" "/*.st[0-9]")
(auto-execute "ties-picture-mode" "/*.pn[0-9]")
(auto-execute "ties-target-mode" "/*.tn[0-9]")
```