```
(IL:RPAQQ IL:ROOMS-CORECOMS
          (;; core rooms code

          (FILE-ENVIRONMENTS IL:ROOMS-CORE)
          (IL:P (EXPORT '(ROOM ROOM-P ROOM-NAME ROOM-PLACEMENTS ROOM-INCLUSIONS ROOM-BACKGROUND
                               ROOM-TTY-PROCESS ROOM-PROPS ROOM-PROP MAKE-ROOM COPY-ROOM DELETE-ROOM
                               RENAME-ROOM ROOM-NAMED ROOM-SORT-FUNCTION))
                (EXPORT '(*CURRENT-ROOM* *POCKET-ROOM-NAME* *ROOM-ENTRY-FUNCTIONS* *ROOM-EXIT-FUNCTIONS*
                               *ROOM-CHANGED-FUNCTIONS*))
                (EXPORT '(PLACEMENT PLACEMENT-P PLACEMENT-WINDOW PLACEMENT-REGION PLACEMENT-SHRUNKEN?
                               PLACEMENT-ICON-POSITION PLACEMENT-PROPS PLACEMENT-PROP MAKE-PLACEMENT
                               COPY-PLACEMENT MOVE-PLACEMENT))
                (EXPORT '(GO-TO-ROOM UPDATE-PLACEMENTS FIND-PLACEMENT ROOM-CHANGED DO-INCLUSIONS RESET))
                (REQUIRE "ROOMS"))
          (IL:COMS

                ;; the room -- a named object

                (IL:STRUCTURES ROOM)
                (IL:VARIABLES *ROOMS* *CURRENT-ROOM*)
                (IL:FUNCTIONS IN-ROOM? MAKE-ROOM COPY-ROOM RENAME-ROOM ROOM-PROP DO-ROOMS ALL-ROOMS
                        ROOM-SORT-FUNCTION ROOM-NAMED DELETE-ROOM))
          (IL:COMS

                ;; placements

                (IL:STRUCTURES PLACEMENT)
                (IL:FUNCTIONS PLACEMENT-PROP MAKE-PLACEMENT COPY-PLACEMENT MOVE-PLACEMENT ADD-PLACEMENT
                        DELETE-PLACEMENT))
          ;; going from one room to another

          (IL:VARIABLES *POCKET-ROOM-NAME* *MONITOR-LOCK* *ROOM-ENTRY-FUNCTIONS* *ROOM-EXIT-FUNCTIONS*)
          (IL:FUNCTIONS GO-TO-ROOM GO-TO-ROOM-PROCESS GO-TO-ROOM-INTERNAL CALL-ENTRY-FUNCTIONS
                  CALL-EXIT-FUNCTIONS UPDATE-PLACEMENTS FIND-PLACEMENT FIND-PLACEMENT-IN-ROOM UPDATE-PLACEMENT
                  PLACE-PLACEMENTS FIND-PLACEMENTS PLACE-PLACEMENT)
          (IL:FUNCTIONS UPDATE-TTY-PROCESS PLACE-TTY-PROCESS)
          ;; other essentials

          (IL:FUNCTIONS FIND-ROOMS-CONTAINING)
          (IL:COMS (IL:VARIABLES *ROOM-CHANGED-FUNCTIONS*)
                  (IL:FUNCTIONS ROOM-CHANGED))
          (IL:FUNCTIONS DO-INCLUSIONS ROOM-INCLUDERS)
          ;; bootstrapping & resetting

          (IL:VARIABLES *RESET-FORMS*)
          (IL:FUNCTIONS RESET)
          (IL:VARIABLES OLD-WHOLESCREEN *SCREEN-CHANGED-FUNCTIONS*)
          (IL:FUNCTIONS AROUNDEXITFN %INTERNALIZE-ALL-PLACEMENTS %INTERNALIZE-PLACEMENTS)
          (IL:GLOBALVARS IL:PROMPTWINDOW IL:AROUNDEXITFNS)
          (EVAL-WHEN (LOAD)
                  (IL:P                                              ; smash system code which moves windows around on reboot so
                                                                     ; we don't fight with it.
                        (PUSHNEW '(IL:CHANGENAME 'IL:\\STARTDISPLAY 'IL:\\MOVE.WINDOWS.ONTO.SCREEN 'IL:NILL)
                                 *RESET-FORMS* :TEST 'EQUAL)))
          ;; random

          (IL:PROP IL:ARGNAMES GO-TO-ROOM)
          (IL:SEDIT-FORMATS DO-INCLUSIONS DO-ROOMS)))



;; core rooms code


(DEFINE-FILE-ENVIRONMENT IL:ROOMS-CORE :COMPILER :COMPILE-FILE
    :PACKAGE "ROOMS"
    :READTABLE "XCL")

(EXPORT '(ROOM ROOM-P ROOM-NAME ROOM-PLACEMENTS ROOM-INCLUSIONS ROOM-BACKGROUND ROOM-TTY-PROCESS ROOM-PROPS
            ROOM-PROP MAKE-ROOM COPY-ROOM DELETE-ROOM RENAME-ROOM ROOM-NAMED ROOM-SORT-FUNCTION))

(EXPORT '(*CURRENT-ROOM* *POCKET-ROOM-NAME* *ROOM-ENTRY-FUNCTIONS* *ROOM-EXIT-FUNCTIONS*
            *ROOM-CHANGED-FUNCTIONS*))

(EXPORT '(PLACEMENT PLACEMENT-P PLACEMENT-WINDOW PLACEMENT-REGION PLACEMENT-SHRUNKEN? PLACEMENT-ICON-POSITION
            PLACEMENT-PROPS PLACEMENT-PROP MAKE-PLACEMENT COPY-PLACEMENT MOVE-PLACEMENT))
```

```
(EXPORT '(GO-TO-ROOM UPDATE-PLACEMENTS FIND-PLACEMENT ROOM-CHANGED DO-INCLUSIONS RESET))

(REQUIRE "ROOMS")
```

;; the room -- a named object

```
(DEFSTRUCT (ROOM (:CONSTRUCTOR MAKE-ROOM-INTERNAL)
                      (:COPIER COPY-ROOM-INTERNAL)
                      (:PRINT-FUNCTION (LAMBDA (ROOM STREAM DEPTH)
                                                (FORMAT STREAM "#<Room ~S>" (ROOM-NAME ROOM)))))
    (NAME NIL :READ-ONLY T)
    (PLACEMENTS NIL :TYPE LIST)
```
  ;; list of PLACEMENT objects
```
    (INCLUSIONS NIL :TYPE LIST)
```
  ;; list of names of included rooms
```
    (BACKGROUND NIL :TYPE BACKGROUND)
```
  ;; how to paint the background
```
    (TTY-PROCESS NIL)
```
  ;; which process has the TTY in this room
```
    (PROPS NIL :TYPE LIST)
```
  ;; property list
```
    )


(DEFVAR *ROOMS* (MAKE-HASH-TABLE :TEST 'EQUAL)
                   "A hash table mapping from room names to rooms.")


(DEFGLOBALVAR *CURRENT-ROOM* NIL
    "The room the user is currently in.")


(DEFUN IN-ROOM? (ROOM)
```
;;; true if ROOM is a sub-room of the current room
```
    (DO-INCLUSIONS (INCLUDED-ROOM *CURRENT-ROOM*)
          (WHEN (EQUAL (ROOM-NAME ROOM)
                        (ROOM-NAME INCLUDED-ROOM))
              (RETURN-FROM DO-INCLUSIONS T))))


(DEFUN MAKE-ROOM (NAME &REST REST-KEYS &KEY PLACEMENTS INCLUSIONS (BACKGROUND NIL BACKGROUND-SPECIFIED?)
                            TTY-PROCESS &ALLOW-OTHER-KEYS)
    ;; check whether a room with this already exists
    (WHEN (ROOM-NAMED NAME)
        (CERROR "Delete existing room named ~S (will close windows)" "A room named ~S already exists" NAME)
        (DELETE-ROOM (ROOM-NAMED NAME)))
    ;; check the types of the placements
    (DOLIST (PLACEMENT PLACEMENTS)
        (CHECK-TYPE PLACEMENT PLACEMENT))
    ;; default the background to contain the name of the room
    (UNLESS BACKGROUND-SPECIFIED?
        (SETQ BACKGROUND '((:TEXT ,NAME))))
    (LET ((ROOM (MAKE-ROOM-INTERNAL :NAME NAME :PLACEMENTS PLACEMENTS :INCLUSIONS INCLUSIONS :BACKGROUND
                          (MAKE-BACKGROUND BACKGROUND)
                          :TTY-PROCESS TTY-PROCESS :PROPS (LET ((PROPS (COPY-LIST REST-KEYS)))
                                                              (DOLIST (KEYWORD '(:PLACEMENTS :INCLUSIONS
                                                                                    :BACKGROUND :TTY-PROCESS))
                                                                  (REMF PROPS KEYWORD))
                                                              PROPS)))))
        (SETF (ROOM-NAMED NAME)
              ROOM)
        (WHEN *CURRENT-ROOM*
            (WHEN (EQUAL NAME (ROOM-NAME *CURRENT-ROOM*))
                (SETQ *CURRENT-ROOM* ROOM))
            (ROOM-CHANGED ROOM :CREATED))
        ROOM))


(DEFUN COPY-ROOM (ROOM NEW-NAME)
    (UPDATE-PLACEMENTS)
    (APPLY 'MAKE-ROOM NEW-NAME :PLACEMENTS (MAPCAR #'COPY-PLACEMENT (ROOM-PLACEMENTS ROOM))
            :INCLUSIONS
            (COPY-LIST (ROOM-INCLUSIONS ROOM))
            :BACKGROUND
            (LET* ((BACKGROUND (COPY-TREE (BACKGROUND-EXTERNAL-FORM (ROOM-BACKGROUND ROOM)))))
```

```
                        (OLD-NAME (ROOM-NAME ROOM))
                        (TEXT (FIND-IF #'(LAMBDA (COMMAND)
                                              (AND (EQ (FIRST COMMAND)
                                                       :TEXT)
                                                   (EQUAL (SECOND COMMAND)
                                                          OLD-NAME)))
                                    BACKGROUND)))
                   (WHEN TEXT
                       (SETF (SECOND TEXT)
                             NEW-NAME))
                   BACKGROUND)
           (COPY-TREE (ROOM-PROPS ROOM))))
```

```
(DEFUN RENAME-ROOM (ROOM NEW-NAME)
    (LET ((OLD-NAME (ROOM-NAME ROOM)))
        (PROG1 (COPY-ROOM ROOM NEW-NAME)
             (DELETE-ROOM ROOM)
             (LET ((SUITE-NAME (FIND-SUITE-CONTAINING OLD-NAME)))

                 ;; if its in a suite, rename it there too

                 (WHEN SUITE-NAME
                     (SETF (SUITE-ROOMS SUITE-NAME)
                           (SUBSTITUTE NEW-NAME OLD-NAME (SUITE-ROOMS SUITE-NAME)
                                  :TEST
                                  'EQUAL))))
                 (DO-ROOMS (ROOM)
                     ;; rename it in inclusions of other rooms

                     (WHEN (MEMBER OLD-NAME (ROOM-INCLUSIONS ROOM)
                                  :TEST
                                  'EQUAL)

                         ;; don't need to call UPDATE-PLACEMENTS as COPY-ROOM has already called it for us.

                         (SETF (ROOM-INCLUSIONS ROOM)
                               (SUBSTITUTE NEW-NAME OLD-NAME (ROOM-INCLUSIONS ROOM)
                                      :TEST
                                      'EQUAL))
                         (ROOM-CHANGED ROOM :EDITED))))))))
```

```
(DEFMACRO ROOM-PROP (ROOM PROP &OPTIONAL (NEW-VALUE NIL NEW-VALUE-SUPPLIED))
    (IF NEW-VALUE-SUPPLIED
        `(SETF (GETF (ROOM-PROPS ,ROOM)
                   ,PROP)
               ,NEW-VALUE)
        `(GETF (ROOM-PROPS ,ROOM)
             ,PROP)))
```

```
(DEFMACRO DO-ROOMS ((ROOM-VAR)
                        &BODY BODY)
```

;;; evaluate BODY once for each room with ROOM-VAR bound to the room.

```
    `(BLOCK DO-ROOMS
        (MAPHASH #'(LAMBDA (,(GENSYM)
                            ,ROOM-VAR)
                         ,@BODY)
                 *ROOMS*)))
```

```
(DEFUN ALL-ROOMS (&OPTIONAL SORTED?)
```

;;; return a list of all rooms. if SORTED? is true, sort them alphabetically by name

```
    (LET ((ALL-ROOMS (WITH-COLLECTION (DO-ROOMS (ROOM)
                                          (COLLECT ROOM)))))
        (IF SORTED?
            (SORT ALL-ROOMS #'ROOM-SORT-FUNCTION)
            ALL-ROOMS)))
```

```
(DEFUN ROOM-SORT-FUNCTION (ROOM-1 ROOM-2)
```

;;; used as the predicate for sorting lists of rooms. we sort alphabetically by the name of the room.

```
    (MACROLET ((STRINGIFY (NAME)
                      `(IF (STRINGP ,NAME)
                           ,NAME
                           (PRINC-TO-STRING ,NAME))))
        (LET ((NAME-1 (ROOM-NAME ROOM-1))
              (NAME-2 (ROOM-NAME ROOM-2)))
            (STRING-LESSP (STRINGIFY NAME-1)
                  (STRINGIFY NAME-2)))))
```

```
(DEFMACRO ROOM-NAMED (NAME)
    `(GETHASH ,NAME *ROOMS*))


(DEFUN DELETE-ROOM (ROOM)
    ;; first close all the windows which only have placements in this room

    (LET ((ONLY-THIS-ROOM (LIST ROOM)))
        (DOLIST (WINDOW (ALL-WINDOWS T))
            (WHEN (EQUAL (FIND-ROOMS-CONTAINING WINDOW)
                            ONLY-THIS-ROOM)
                (UN-HIDE-WINDOW WINDOW)
                (CLOSE-WINDOW (IF (SHRUNKEN? WINDOW)
                                    (WINDOW-ICON WINDOW)
                                    WINDOW)))))
        (WHEN (DO-ROOMS (RM)
                    (WHEN (EQ ROOM RM)
                        (RETURN-FROM DO-ROOMS T)))
    ;; if it's in the name table, remove it.  this is so deleting an un-named room (like the Overview) doesn't cause a room named "Overview" to also
    ;; disappear.

            (REMHASH (ROOM-NAME ROOM)
                    *ROOMS*))
    ;; tell the world we've deleted it

    (ROOM-CHANGED ROOM :DELETED))


;; placements


(DEFSTRUCT (PLACEMENT (:CONSTRUCTOR MAKE-PLACEMENT-INTERNAL)
                            (:COPIER COPY-PLACEMENT-INTERNAL))
    WINDOW
    REGION
    SHRUNKEN?
    ICON-POSITION
    PROPS)


(DEFMACRO PLACEMENT-PROP (PLACEMENT PROP &OPTIONAL (NEW-VALUE NIL NEW-VALUE-SUPPLIED))
    (IF NEW-VALUE-SUPPLIED
        `(SETF (GETF (PLACEMENT-PROPS ,PLACEMENT)
                    ,PROP)
            ,NEW-VALUE)
        `(GETF (PLACEMENT-PROPS ,PLACEMENT)
            ,PROP)))


(DEFUN MAKE-PLACEMENT (WINDOW)
    (LET ((PLACEMENT (MAKE-PLACEMENT-INTERNAL :WINDOW WINDOW)))
        (UPDATE-PLACEMENT PLACEMENT)
        PLACEMENT))


(DEFUN COPY-PLACEMENT (PLACEMENT)
    ;; make sure PROPS gets copied.  it is not important that REGION & ICON-POSITION are copied, but seems safer.

    (MAKE-PLACEMENT-INTERNAL :WINDOW (PLACEMENT-WINDOW PLACEMENT)
            :REGION
            (COPY-REGION (PLACEMENT-REGION PLACEMENT))
            :SHRUNKEN?
            (PLACEMENT-SHRUNKEN? PLACEMENT)
            :ICON-POSITION
            (COPY-TREE (PLACEMENT-ICON-POSITION PLACEMENT))
            :PROPS
            (COPY-TREE (PLACEMENT-PROPS PLACEMENT))))


(DEFUN MOVE-PLACEMENT (PLACEMENT FROM-ROOM TO-ROOM &OPTIONAL COPY?)
    (ADD-PLACEMENT (COPY-PLACEMENT PLACEMENT)
            TO-ROOM)
    (UNLESS COPY?
        (DELETE-PLACEMENT PLACEMENT FROM-ROOM)
        (LET* ((WINDOW (PLACEMENT-WINDOW PLACEMENT))
                (INHERITED (FIND-PLACEMENT WINDOW)))
            (HIDE-WINDOW WINDOW)
            (WHEN INHERITED (PLACE-PLACEMENT INHERITED)))))


(DEFUN ADD-PLACEMENT (PLACEMENT ROOM)

;;; add PLACEMENT to ROOM's placements.  does not update screen.
    ;; first delete any old placements for same window

    (SETF (ROOM-PLACEMENTS ROOM)
        (DELETE (PLACEMENT-WINDOW PLACEMENT)
                (ROOM-PLACEMENTS ROOM)
```

```
                    :TEST
                    'EQ :KEY #'PLACEMENT-WINDOW))
```
;; add it
```
(PUSH PLACEMENT (ROOM-PLACEMENTS ROOM))
```
;; notify system that ROOM has changed.
```
(ROOM-CHANGED ROOM :PLACEMENTS))
```


```
(DEFUN DELETE-PLACEMENT (PLACEMENT ROOM)
```
  ;; delete PLACEMENT from ROOM. does not remove placement from screen.
```
(SETF (ROOM-PLACEMENTS ROOM)
      (DELETE (PLACEMENT-WINDOW PLACEMENT)
              (ROOM-PLACEMENTS ROOM)
              :TEST
              'EQ :KEY #'PLACEMENT-WINDOW))
```
  ;; notify system that ROOM has changed.
```
(ROOM-CHANGED ROOM :PLACEMENTS))
```


;; going from one room to another


```
(DEFGLOBALVAR *POCKET-ROOM-NAME* NIL
   "The name of the room to be the pockets or NIL.")
```


```
(DEFGLOBALVAR *MONITOR-LOCK*)
```


```
(DEFVAR *ROOM-ENTRY-FUNCTIONS* NIL
   "A list of functions to be called before a room is entered")
```


```
(DEFVAR *ROOM-EXIT-FUNCTIONS* NIL
   "A list of functions to be called before a room is left")
```


```
(DEFUN GO-TO-ROOM (&REST ARGS)
```

;;; skip to GO-TO-ROOM-INTERNAL for details...

  ;; can't run under mouse, as mouse switches TTY around. have to spawn our own process, let the mouse return the TTY, then we'll be run.
```
(CHECK-TYPE (FIRST ARGS)
        ROOM)
(IL:RESETVAR IL:\\PROC.RUN.NEXT.FLG T
```
  ;; ensure that we'll be the next process run when the mouse blocks.
```
(IL:ADD.PROCESS '(GO-TO-ROOM-PROCESS ',ARGS)
        'IL:NAME "Go To Room")))
```


```
(DEFUN GO-TO-ROOM-PROCESS (ARGS)
   (LET ((OLD-CURSOR (IL:CURSOR)))
        (UNWIND-PROTECT
            (IF (IL:OBTAIN.MONITORLOCK *MONITOR-LOCK* T)
                (PROGN (IL:CURSOR IL:WAITINGCURSOR)
                       (IL:\\CARET.DOWN NIL IL:MAX.FIXP)
                       (APPLY 'GO-TO-ROOM-INTERNAL ARGS))
                (NOTIFY-USER "Can't!  Rooms is busy."))
            (IL:RELEASE.MONITORLOCK *MONITOR-LOCK*)
            (IL:CURSOR OLD-CURSOR)
            (IL:CARET T))))
```


```
(DEFUN GO-TO-ROOM-INTERNAL (ROOM &KEY NO-UPDATE BAGGAGE)
   (CHECK-TYPE ROOM ROOM)
```

;;; Leave the current room & enter ROOM. BAGGAGE is a list of additional placements to be placed in ROOM.
  ;; call exit hooks on current room
```
(CALL-EXIT-FUNCTIONS *CURRENT-ROOM*)
(UNLESS NO-UPDATE
```
    ;; update the current room per the screen
```
(UPDATE-PLACEMENTS *CURRENT-ROOM*))
```
  ;; note which process has the keyboard
```
(UPDATE-TTY-PROCESS *CURRENT-ROOM*)
```
  ;; clear the screen
```
(HIDE-ALL-WINDOWS)
(UNWIND-PROTECT
     (PROGN  ;; paint the background
```

```
                    (PAINT-BACKGROUND ROOM *SCREEN-BITMAP*)
```
            ;; call entry hooks
```
                    (CALL-ENTRY-FUNCTIONS ROOM))
```
        ;; set *CURRENT-ROOM*.
```
            (SETQ *CURRENT-ROOM* ROOM))
```
    ;; place placements from ROOM -- inherited & direct
```
        (PLACE-PLACEMENTS ROOM BAGGAGE)
```
    ;; place the caret
```
        (PLACE-TTY-PROCESS ROOM))
```

```
(DEFUN CALL-ENTRY-FUNCTIONS (ROOM)
```
    ;; first call global entry functions
```
        (DOLIST (FN *ROOM-ENTRY-FUNCTIONS*)
            (FUNCALL FN ROOM))
```
    ;; then call inherited entry functions
```
        (DO-INCLUSIONS (SUB-ROOM ROOM)
            (DOLIST (FN (ROOM-PROP SUB-ROOM :BEFORE-ENTRY-FUNCTIONS))
                (FUNCALL FN ROOM))))
```

```
(DEFUN CALL-EXIT-FUNCTIONS (ROOM)
```
    ;; first call global room exit functions
```
        (DOLIST (FN *ROOM-EXIT-FUNCTIONS*)
            (FUNCALL FN ROOM))
```
    ;; then call inherited functions on ROOM
```
        (DO-INCLUSIONS (SUB-ROOM ROOM)
            (DOLIST (FN (ROOM-PROP SUB-ROOM :BEFORE-EXIT-FUNCTIONS))
                (FUNCALL FN ROOM))))
```

```
(DEFUN UPDATE-PLACEMENTS (&OPTIONAL (FOR-ROOM *CURRENT-ROOM*))
```

;;; called when leaving a room to update it's placements

;;; returns the new list of placements

```
    (LET ((NEW-PLACEMENTS NIL)
          (CHANGED-ROOMS NIL)
          (OLD-PLACEMENTS (ROOM-PLACEMENTS FOR-ROOM))
          (ALL-WINDOWS (ALL-WINDOWS)))
        (DOLIST (WINDOW ALL-WINDOWS)
            (MULTIPLE-VALUE-BIND (PLACEMENT IN-ROOM)
                (FIND-PLACEMENT WINDOW FOR-ROOM)
                (UNLESS PLACEMENT
```
                    ;; new window in this room - make a placement
```
                    (SETQ PLACEMENT (MAKE-PLACEMENT WINDOW))
                    (SETQ IN-ROOM FOR-ROOM)
```
                    ;; note change to this room
```
                    (PUSHNEW FOR-ROOM CHANGED-ROOMS :TEST 'EQ))
```
                ;; collect placements in this room in top to bottom order.
```
                (WHEN (EQ IN-ROOM FOR-ROOM)
                    (PUSH PLACEMENT NEW-PLACEMENTS))
```
                ;; update the placement
```
                (WHEN (UPDATE-PLACEMENT PLACEMENT)
```
                    ;; placement has changed - note it
```
                    (PUSHNEW IN-ROOM CHANGED-ROOMS :TEST 'EQ))))
        (DOLIST (PLACEMENT (FIND-PLACEMENTS FOR-ROOM))
            (UNLESS (MEMBER (PLACEMENT-WINDOW PLACEMENT)
                            ALL-WINDOWS :TEST 'EQ)
```
                ;; it's a window that's been closed
```
                (DO-INCLUSIONS (ROOM FOR-ROOM)
                    (WHEN (MEMBER PLACEMENT (ROOM-PLACEMENTS ROOM)
                                  :TEST
                                  'EQ)
```
                        ;; delete its placement
```
                        (UNLESS (EQ ROOM FOR-ROOM)
```
                            ;; unless we'll delete it below anyway
```
                            (DELETE-PLACEMENT PLACEMENT ROOM))
```
                        ;; note that this room has changed
```

```
                                    (PUSHNEW ROOM CHANGED-ROOMS :TEST 'EQ)
                                    (RETURN-FROM DO-INCLUSIONS)))))
                (UNLESS (EQUAL NEW-PLACEMENTS OLD-PLACEMENTS)

                    ;; check if occlusion order of placements has changed

                    (PUSHNEW FOR-ROOM CHANGED-ROOMS :TEST 'EQ))
                (SETF (ROOM-PLACEMENTS FOR-ROOM)
                      NEW-PLACEMENTS)
                (DOLIST (ROOM CHANGED-ROOMS)
                    (ROOM-CHANGED ROOM :PLACEMENTS))
                T))


(DEFUN FIND-PLACEMENT (WINDOW &OPTIONAL (FROM-ROOM *CURRENT-ROOM*))
```

;;; returns the placement which caused WINDOW to be in ROOM.

;;; does a breadth-first search through ROOM & its inclusions for a placement containing WINDOW. second value is room placement was found in.

```
        (DO-INCLUSIONS (ROOM FROM-ROOM)
            (LET ((PLACEMENT (FIND-PLACEMENT-IN-ROOM WINDOW ROOM)))
                (WHEN PLACEMENT
                    (RETURN-FROM FIND-PLACEMENT (VALUES PLACEMENT ROOM))))))


(DEFMACRO FIND-PLACEMENT-IN-ROOM (WINDOW ROOM)
    `(LET ((WINDOW ,WINDOW))
        (DOLIST (PLACEMENT (ROOM-PLACEMENTS ,ROOM))
            (WHEN (EQ (PLACEMENT-WINDOW PLACEMENT)
                      WINDOW)
                (RETURN PLACEMENT)))))


(DEFUN UPDATE-PLACEMENT (PLACEMENT)
```

;;; called when leaving a room on each placement in the room. returns true if placement has changed since the last time it was updated.

```
    (LET* ((WINDOW (PLACEMENT-WINDOW PLACEMENT))
           (ICON-POSITION (ICON-POSITION WINDOW))
           (REGION (WINDOW-REGION WINDOW))
           (SHRUNKEN? (SHRUNKEN? WINDOW))
           (CHANGED? NIL))
        (UNLESS (EQUAL ICON-POSITION (PLACEMENT-ICON-POSITION PLACEMENT))
            (SETF (PLACEMENT-ICON-POSITION PLACEMENT)
                  (COPY-TREE ICON-POSITION))
            (SETQ CHANGED? T))
        (UNLESS (EQUAL REGION (PLACEMENT-REGION PLACEMENT))
            (SETF (PLACEMENT-REGION PLACEMENT)
                  (COPY-REGION REGION))
            (SETQ CHANGED? T))
        (UNLESS (EQ SHRUNKEN? (PLACEMENT-SHRUNKEN? PLACEMENT))
            (SETF (PLACEMENT-SHRUNKEN? PLACEMENT)
                  SHRUNKEN?)
            (SETQ CHANGED? T))

        ;; call the user hook

        (LET ((WINDOW-TYPE (WINDOW-TYPE WINDOW T)))
            (WHEN WINDOW-TYPE
                (LET ((UPDATER (WINDOW-TYPE-UPDATER WINDOW-TYPE)))
                    (WHEN UPDATER
                        (FUNCALL (WINDOW-TYPE-UPDATER WINDOW-TYPE)
                                 PLACEMENT)))))
        CHANGED?))


(DEFUN PLACE-PLACEMENTS (ROOM &OPTIONAL BAGGAGE)
    (DOLIST (PLACEMENT (FIND-PLACEMENTS ROOM))
        (PLACE-PLACEMENT PLACEMENT))
    (DOLIST (PLACEMENT BAGGAGE)
        (PLACE-PLACEMENT PLACEMENT)))


(DEFUN FIND-PLACEMENTS (ROOM)
```

;;; returns the list of placements to be displayed in room, ordered in bottom first (i.e. the order they should be displayed in)

```
    (LET (PLACEMENTS)
        (DO-INCLUSIONS (INCLUSION ROOM)
            (DOLIST (PLACEMENT (ROOM-PLACEMENTS INCLUSION))

                ;; save one placement for each window on the way down

                ;; optimization: this rather convoluted piece of code is used rather than (pushnew placement placements :key
                ;; #'placement-window) because pushnew compiles into something really slow in XCL.

                (LET ((WINDOW (PLACEMENT-WINDOW PLACEMENT)))
                    (UNLESS (DOLIST (PLACEMENT PLACEMENTS)
                                (WHEN (EQ (PLACEMENT-WINDOW PLACEMENT)
                                          WINDOW)
```

```
                                              (RETURN T)))
                              (PUSH PLACEMENT PLACEMENTS)))))
          PLACEMENTS))
```

(DEFUN **PLACE-PLACEMENT** (PLACEMENT)

;;; Called on each placement in a room when it's visited to place PLACEMENT's window per the rest of PLACEMENT.

;;; This will probably require a lot of work in a different window system.

```
    (LET* ((WINDOW (PLACEMENT-WINDOW PLACEMENT))
```
            ;; we copy as window system sometimes seems to smash these
```
           (PLACEMENT-REGION (COPY-REGION (PLACEMENT-REGION PLACEMENT)))
           (PLACEMENT-ICON-POSITION (COPY-TREE (PLACEMENT-ICON-POSITION PLACEMENT)))
           (WINDOW-REGION (WINDOW-REGION WINDOW))
           (WINDOW-ICON (WINDOW-ICON WINDOW))
           (WINDOW-TYPE (WINDOW-TYPE WINDOW T)))
       (WHEN (OR (IL:OPENWP WINDOW)
                 (AND WINDOW-ICON (IL:OPENWP WINDOW-ICON)))
```
             ;; if it's been closed & we ignore it
```
          (UN-HIDE-WINDOW WINDOW)
          (COND
             ((PLACEMENT-SHRUNKEN? PLACEMENT)
```
                ;; ensure the expansion is placed correctly
```
              (UNLESS (EQUAL PLACEMENT-REGION WINDOW-REGION)
                  (SHAPE-WINDOW WINDOW PLACEMENT-REGION :CURRENT-REGION WINDOW-REGION :NO-SHAPE
                      (AND WINDOW-TYPE (WINDOW-TYPE-PROP WINDOW-TYPE :NO-SHAPE))))
```
                ;; place the icon
```
              (COND
                 ((SHRUNKEN? WINDOW)
                  (UNLESS (EQUAL (WINDOW-POSITION WINDOW-ICON)
                                 PLACEMENT-ICON-POSITION)
                     (MOVE-WINDOW WINDOW-ICON PLACEMENT-ICON-POSITION)
                     (OPEN-WINDOW WINDOW-ICON)))
                 (T (MOVE-WINDOW (SHRINK-WINDOW WINDOW PLACEMENT-ICON-POSITION)
                        PLACEMENT-ICON-POSITION))))
             (T (WHEN PLACEMENT-ICON-POSITION
```
                   ;; ensure the icon is placed correctly
```
                  (UNLESS WINDOW-ICON
                      (SETQ WINDOW-ICON (SHRINK-WINDOW WINDOW PLACEMENT-ICON-POSITION)))
                  (MOVE-WINDOW WINDOW-ICON PLACEMENT-ICON-POSITION)
                  (WHEN (AND (IL:OPENWP WINDOW-ICON)
                             (NOT (SHRUNKEN? WINDOW)))
```
                      ;; we opened the icon by moving it
```
                     (IL:\\CLOSEW1 WINDOW-ICON)))
```
                ;; place the window
```
                (WHEN (SHRUNKEN? WINDOW)
                    (EXPAND-WINDOW WINDOW))
                (UNLESS (EQUAL PLACEMENT-REGION WINDOW-REGION)
                    (SHAPE-WINDOW WINDOW PLACEMENT-REGION :CURRENT-REGION WINDOW-REGION :NO-SHAPE
                        (AND WINDOW-TYPE (WINDOW-TYPE-PROP WINDOW-TYPE :NO-SHAPE))))
                (OPEN-WINDOW WINDOW)
                (UNLESS PLACEMENT-ICON-POSITION (DELETE-WINDOW-ICON WINDOW))))
```
             ;; call the user hook
```
          (WHEN (AND WINDOW-TYPE (WINDOW-TYPE-PLACER WINDOW-TYPE))
              (FUNCALL (WINDOW-TYPE-PLACER WINDOW-TYPE)
                      PLACEMENT)))))
```

(DEFUN **UPDATE-TTY-PROCESS** (ROOM)

;;; update ROOM's notion of which process has the keyboard.

```
    (SETF (ROOM-TTY-PROCESS ROOM)
          (IL:TTY.PROCESS)))
```

(DEFUN **PLACE-TTY-PROCESS** (ROOM)

;;; place the keyboard per ROOM's TTY-PROCESS field

```
    (LET ((PROCESS (ROOM-TTY-PROCESS ROOM)))
        (IL:TTY.PROCESS (IF (IL:PROCESSP PROCESS)
                            PROCESS
```
                             ;; if no process specified, or the specified process is dead, then we give the TTY to the MOUSE process
```
                            (IL:FIND.PROCESS 'IL:MOUSE)))))
```

;; other essentials

(DEFUN **FIND-ROOMS-CONTAINING** (WINDOW)

;;; return a list of all rooms which directly contain a placement for WINDOW

```
    (LET ((ROOMS))
         (DO-ROOMS (ROOM)
              (WHEN (FIND-PLACEMENT-IN-ROOM WINDOW ROOM)
                   (PUSH ROOM ROOMS)))
         ;; we need a general way of handling un-named rooms, but as there is only one now, we can just special case it.
         (WHEN (FIND-PLACEMENT-IN-ROOM WINDOW *OVERVIEW-ROOM*)
               (PUSH *OVERVIEW-ROOM* ROOMS))
         ROOMS))
```


(DEFGLOBALVAR ***ROOM-CHANGED-FUNCTIONS*** NIL)

(DEFUN **ROOM-CHANGED** (ROOM REASON)

;;; called when we notice a room has changed to ensure display is up to date.

```
    (ECASE REASON
        ((:EDITED :CREATED :DELETED) (WHEN (IN-ROOM? ROOM)
                                          ;; if we're in this room, redisplay whole screen
                                          ;; note: we depend upon our caller to update placements
                                          (IL:WITH.MONITOR *MONITOR-LOCK* (GO-TO-ROOM-INTERNAL *CURRENT-ROOM*
                                                                                               :NO-UPDATE T))))
        (:PLACEMENTS  ;; we presume our caller & the hooks handle these cases
                     ))
    ;; call hooks
    (DOLIST (FN *ROOM-CHANGED-FUNCTIONS*)
        (FUNCALL FN ROOM REASON)))
```


(DEFMACRO **DO-INCLUSIONS** ((ROOM-VAR ROOM-FORM)
                              &BODY BODY)

;;; descend breadth-first, left to right down the inclusions of a room, performing BODY with ROOM-VAR bound to each room.

```
    `(LET* ((,ROOM-VAR ,ROOM-FORM)
            ($ROOMS$ (LIST ,ROOM-VAR))
            ($QUEUE-HEAD$ $ROOMS$)
            ($QUEUE-TAIL$ $QUEUE-HEAD$)
            ($POCKET-ROOM-NAME$ *POCKET-ROOM-NAME*)
            $INCLUSIONS$ $INCLUSION$)
        (BLOCK DO-INCLUSIONS
            (TAGBODY $LOOP$ ,@BODY (SETQ $INCLUSIONS$ (ROOM-INCLUSIONS ,ROOM-VAR))
                 (UNLESS (LISTP $INCLUSIONS$)
                         (RETURN-FROM DO-INCLUSIONS))
                 (DOLIST (INCLUDED-ROOM-NAME $INCLUSIONS$)
                     (SETQ $INCLUSION$ (ROOM-NAMED INCLUDED-ROOM-NAME))
                     (WHEN (AND $INCLUSION$ (NOT (MEMBER $INCLUSION$ $ROOMS$ :TEST #'EQ)))
                           (RPLACD $QUEUE-TAIL$ (SETQ $QUEUE-TAIL$ (LIST $INCLUSION$)))))
                 (POP $QUEUE-HEAD$)
                 (IF $QUEUE-HEAD$
                     (SETQ ,ROOM-VAR (FIRST $QUEUE-HEAD$))
                     (IF (AND $POCKET-ROOM-NAME$ (SETQ ,ROOM-VAR (ROOM-NAMED $POCKET-ROOM-NAME$))
                             (NOT (MEMBER ,ROOM-VAR $ROOMS$ :TEST #'EQ)))
                         (SETQ $POCKET-ROOM-NAME$ NIL)
                         (RETURN-FROM DO-INCLUSIONS)))
                 (GO $LOOP$)))))
```


(DEFUN **ROOM-INCLUDERS** (ROOM &OPTIONAL SORTED?)

;;; returns the list of rooms which include ROOM.

;;; note that every room implicitly includes itself. the motivation for this is that most code which wants to map over includers also wants the root.

```
    (IF (EQUAL (ROOM-NAME ROOM)
               *POCKET-ROOM-NAME*)
        ;; special case: all rooms include the pocket room
        (ALL-ROOMS SORTED?)
        (DO* ((INCLUDERS NIL)                                  ; list of included rooms
              (QUEUE (LIST ROOM))                              ; list of rooms to examine
              (INCLUDER ROOM (POP QUEUE))                      ; room being examined
              (INCLUDER-NAME (ROOM-NAME INCLUDER)
                    (ROOM-NAME INCLUDER)))
             ((NULL QUEUE)
```

```
                    (IF SORTED?
                        (SORT INCLUDERS #'ROOM-SORT-FUNCTION)
                        INCLUDERS))
                (UNLESS (MEMBER INCLUDER INCLUDERS :TEST 'EQ)
                    (PUSH INCLUDER INCLUDERS)
                    (DO-ROOMS (ROOM)
                        (LET ((INCLUSIONS (ROOM-INCLUSIONS ROOM)))
                            (WHEN (AND (LISTP INCLUSIONS)
                                       (MEMBER INCLUDER-NAME INCLUSIONS :TEST 'EQUAL))
                                (PUSHNEW ROOM QUEUE :TEST 'EQ))))))))))
```

;; bootstrapping & resetting


```
(DEFVAR *RESET-FORMS* NIL
    "List of forms to be EVALled when Rooms is reset.")
```


```
(DEFUN RESET ()
```
   ;; delete all existing rooms

   `(CLRHASH *ROOMS*)`

   ;; bootstrap *CURRENT-ROOM*

```
(SETQ *CURRENT-ROOM* NIL)
(SETQ *POCKET-ROOM-NAME* "Pockets")
(MAKE-ROOM *POCKET-ROOM-NAME* :PLACEMENTS
```
            ;; put promptwindow in pockets

```
            (LIST (MAKE-PLACEMENT IL:PROMPTWINDOW))
            :BACKGROUND
            (COPY-TREE '((:WHOLE-SCREEN (:EVAL IL:WINDOWBACKGROUNDSHADE)))))
(SETQ *CURRENT-ROOM* (MAKE-ROOM "Original"))
(SETQ *MONITOR-LOCK* (IL:CREATE.MONITORLOCK "Rooms"))
(IL:WITH.MONITOR *MONITOR-LOCK* (GO-TO-ROOM-INTERNAL *CURRENT-ROOM*))
```
   ;; install our aroundexitfn last so it gets called before greet

```
(UNLESS (MEMBER 'AROUNDEXITFN IL:AROUNDEXITFNS)
    (SETQ IL:AROUNDEXITFNS (NCONC IL:AROUNDEXITFNS (LIST 'AROUNDEXITFN))))
```
   ;; do reset forms

```
(DOLIST (FORM *RESET-FORMS*)
    (EVAL FORM))
```
   ;; may have lost some windows...

   `(CHECK-LOST-WINDOWS))`

`(DEFGLOBALVAR OLD-WHOLESCREEN (COPY-REGION IL:WHOLESCREEN))`

`(DEFGLOBALVAR *SCREEN-CHANGED-FUNCTIONS* (LIST '%INTERNALIZE-ALL-PLACEMENTS))`


```
(DEFUN AROUNDEXITFN (EVENT)
    (CASE EVENT
        ((IL:BEFORESAVEVM IL:BEFORELOGOUT IL:BEFORESYSOUT IL:BEFOREMAKESYS) )
        ((IL:AFTERSAVEVM IL:AFTERLOGOUT IL:AFTERSYSOUT IL:AFTERMAKESYS) (UNLESS (EQUAL IL:WHOLESCREEN
                                                                                       OLD-WHOLESCREEN)
                                                                            (DOLIST #'*SCREEN-CHANGED-FUNCTIONS*
                                                                                (FUNCALL FUNCTION))
                                                                            (SETQ OLD-WHOLESCREEN (COPY-REGION
                                                                                                    IL:WHOLESCREEN
                                                                                                    )))))))
```


```
(DEFUN %INTERNALIZE-ALL-PLACEMENTS ()
```
;;; called when we re-boot on different sized screen. re-scales the placement regions & icon-positions of all placements.

```
    (LET ((OLD-SCREEN-WIDTH (REGION-WIDTH OLD-WHOLESCREEN))
          (OLD-SCREEN-HEIGHT (REGION-HEIGHT OLD-WHOLESCREEN)))
        (UPDATE-PLACEMENTS)
        (DO-ROOMS (ROOM)
```
                ;; do all the named rooms

```
                (%INTERNALIZE-PLACEMENTS ROOM OLD-SCREEN-WIDTH OLD-SCREEN-HEIGHT)
                (ROOM-CHANGED ROOM :PLACEMENTS))
```
        ;; redisplay the current room.

```
        (IL:PROCESS.RESULT (GO-TO-ROOM *CURRENT-ROOM* :NO-UPDATE T)
            T)))
```


```
(DEFUN %INTERNALIZE-PLACEMENTS (ROOM OLD-SCREEN-WIDTH OLD-SCREEN-HEIGHT)
    (DOLIST (PLACEMENT (ROOM-PLACEMENTS ROOM))
```

```
        ;; re-scale placements to new size of screen

        (LET ((REGION (PLACEMENT-REGION PLACEMENT)))
             (SETF (PLACEMENT-REGION PLACEMENT)
                   (INTERNALIZE-REGION (MAKE-REGION :LEFT (EXTERNALIZE-COORDINATE (REGION-LEFT REGION)
                                                                                  OLD-SCREEN-WIDTH)
                                                    :BOTTOM
                                                    (EXTERNALIZE-COORDINATE (REGION-BOTTOM REGION)
                                                          OLD-SCREEN-HEIGHT)
                                                    :WIDTH
                                                    (EXTERNALIZE-COORDINATE (REGION-WIDTH REGION)
                                                          OLD-SCREEN-WIDTH)
                                                    :HEIGHT
                                                    (EXTERNALIZE-COORDINATE (REGION-HEIGHT REGION)
                                                          OLD-SCREEN-HEIGHT)))))
        (LET ((POSITION (PLACEMENT-ICON-POSITION PLACEMENT)))
             (WHEN POSITION
                (SETF (PLACEMENT-ICON-POSITION PLACEMENT)
                      (INTERNALIZE-POSITION (MAKE-POSITION (EXTERNALIZE-COORDINATE (POSITION-X POSITION)
                                                                 OLD-SCREEN-WIDTH)
                                                          (EXTERNALIZE-COORDINATE (POSITION-Y POSITION)
                                                                OLD-SCREEN-HEIGHT)))))))))

(IL:DECLARE\: IL:DOEVAL@COMPILE IL:DONTCOPY

(IL:GLOBALVARS IL:PROMPTWINDOW IL:AROUNDEXITFNS)
)

(EVAL-WHEN (LOAD)

;; smash system code which moves windows around on reboot so we don't fight with it.

(PUSHNEW '(IL:CHANGENAME 'IL:\\STARTDISPLAY 'IL:\\MOVE.WINDOWS.ONTO.SCREEN 'IL:NILL)
      *RESET-FORMS* :TEST 'EQUAL)
)

;; random

(IL:PUTPROPS GO-TO-ROOM IL:ARGNAMES (ROOM &KEY NO-UPDATE BAGGAGE))


(SEDIT:DEF-LIST-FORMAT DO-INCLUSIONS :INDENT (1)
    :ARGS (:KEYWORD :BINDING NIL)
    :SUBLISTS (2))


(SEDIT:DEF-LIST-FORMAT DO-ROOMS :INDENT (1)
    :ARGS (:KEYWORD :BINDING NIL)
    :SUBLISTS (2))

(IL:PUTPROPS IL:ROOMS-CORE IL:COPYRIGHT ("Venue & Xerox Corporation" 1987 1988 1990 2020))
```

FUNCTION INDEX

VARIABLE INDEX

MACRO INDEX

SEDIT-FORMAT INDEX

STRUCTURE INDEX

PROPERTY INDEX

FILE-ENVIRONMENT INDEX