File created: 5-Dec-2020 16:35:25 **{DSK}<Users>arunwelch>SKYDRIVE>DOCUMENTS>UNIX>LISP>LDE>ROOMS>MEDLEY-35>ROOMS-SUITES.;2**

previous date: 17-Aug-90 13:29:14 {DSK}<Users>arunwelch>SKYDRIVE>DOCUMENTS>UNIX>LISP>LDE>ROOMS>MEDLEY-35>ROOMS-SUITES.;1

Read Table: XCL

Package: ROOMS

Format: XCCS

```
(IL:RPAQQ IL:ROOMS-SUITESCOMS
          ((FILE-ENVIRONMENTS IL:ROOMS-SUITES)
           (IL:P (EXPORT '(*SUITE-FILE-TYPE* *SUITE-DIRECTORIES* DEFSUITE))
                 (REQUIRE "ROOMS"))
           (IL:DEFINE-TYPES IL:SUITES)
           (IL:VARIABLES *DEFSUITE-VERSION* *SUITES* *SUITE-DIRECTORIES* *SUITE-FILE-TYPE*)
           (IL:P (PUSHNEW '(CLRHASH *SUITES*)
                        *RESET-FORMS* :TEST 'EQUAL))
           ;; loading suites
           (IL:FUNCTIONS DEFSUITE MAKE-SUITE SUITE-ROOMS INSTALL-SUITE INSTALL-SUITE-BODY INSTALL-ROOM
                  INSTALL-PLACEMENT)
           (IL:SEDIT-FORMATS DEFSUITE)
           ;; deleting suites
           (IL:FUNCTIONS DELETE-SUITE)
           ;; constructing suites
           (IL:FUNCTIONS DUMP-SUITE UPDATE-SUITE CONSTRUCT-DEFSUITE ALL-WINDOWS-IN-ROOMS CONSTRUCT-WINDOWS
                  CONSTRUCT-FILES CONSTRUCT-PLACEMENTS)
           (IL:FUNCTIONS ROOMS-NOT-IN-ANY-SUITE FIND-SUITE-CONTAINING AUGMENT-SUITE DELETE-ROOM-FROM-SUITE)
           ;; interactive code
           (IL:VARIABLES *SUITE-MENU-ITEMS* *SUITE-MENU*)
           (IL:FUNCTIONS SUITE-MENU INTERACTIVE-LOAD-SUITE INTERACTIVE-DUMP-SUITE INTERACTIVE-MAKE-SUITE
                  INTERACTIVE-UPDATE-SUITE INTERACTIVE-DELETE-SUITE INTERACTIVE-SHOW-SUITE
                  INTERACTIVE-AUGMENT-SUITE INTERACTIVE-DELETE-ROOM-FROM-SUITE SELECT-SUITE
                  PROMPT-FOR-SUITE-NAME)))

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

(EXPORT '(*SUITE-FILE-TYPE* *SUITE-DIRECTORIES* DEFSUITE))

(REQUIRE "ROOMS")

(DEF-DEFINE-TYPE IL:SUITES "Room Suites")

(DEFCONSTANT *DEFSUITE-VERSION* 1)

(DEFGLOBALVAR *SUITES* (MAKE-HASH-TABLE :TEST 'EQUAL))

(DEFPARAMETER *SUITE-DIRECTORIES*
   ;; initially just the connected directory
   (LIST T))

(DEFVAR *SUITE-FILE-TYPE* "SUITE")

(PUSHNEW '(CLRHASH *SUITES*)
       *RESET-FORMS* :TEST 'EQUAL)

;; loading suites

(DEFDEFINER (DEFSUITE (:UNDEFINER (LAMBDA (NAME)
                                             (REMHASH NAME *SUITES*))))
    IL:SUITES (NAME &BODY BODY)
   `(INSTALL-SUITE ',NAME ',BODY))

(DEFUN MAKE-SUITE (SUITE-NAME ROOM-NAMES)
   (WHEN (SUITE-ROOMS SUITE-NAME)
       (CERROR "Delete all rooms in existing suite named ~S" "Attempt to redefine suite ~S" SUITE-NAME)
       (DELETE-SUITE SUITE-NAME T))
   (SETF (SUITE-ROOMS SUITE-NAME)
         ROOM-NAMES)
   SUITE-NAME)


(DEFMACRO SUITE-ROOMS (SUITE-NAME)
   `(GETHASH ,SUITE-NAME *SUITES*))


(DEFUN INSTALL-SUITE (SUITE-NAME SUITE-BODY)

;;; do the work of DEFSUITE.

   ;; first check for suite redefinition.  MAKE-SUITE will do this for us, but by checking first we assure things will be left consistent should the user abort.

   (WHEN (SUITE-ROOMS SUITE-NAME)
       (CERROR "Delete all rooms in existing suite named ~S" "Attempt to redefine suite ~S" SUITE-NAME)
       (DELETE-SUITE SUITE-NAME T))
   (LET ((ROOM-NAMES (WITH-COLLECTION (DOLIST (SPEC SUITE-BODY)
                                          (CASE (FIRST SPEC)
                                              (:ROOM (COLLECT (SECOND SPEC))))))))
        (DOLIST (ROOM-NAME ROOM-NAMES)

            ;; check for room redefinitions.  MAKE-ROOM will do this again, but checking here leaves us in a much more consitstent state should
            ;; the user choose to abort rather than continue.

            (WHEN (ROOM-NAMED ROOM-NAME)
                (CERROR "Delete existing room named ~S (will close windows)" "A room named ~S already exists"
                       ROOM-NAME)
                (DELETE-ROOM (ROOM-NAMED ROOM-NAME))))

        ;; make the rooms

        (INSTALL-SUITE-BODY SUITE-BODY)

        ;; remember what rooms were in this suite

        (MAKE-SUITE SUITE-NAME ROOM-NAMES)
        (CHECK-LOST-WINDOWS)))


(DEFUN INSTALL-SUITE-BODY (SUITE-BODY)
   (IL:WITH.MONITOR *MONITOR-LOCK*
       (LET ((WINDOWS (MAKE-HASH-TABLE))
             (VERSION NIL))

            ;; first load files & parse version

            (DOLIST (SPEC SUITE-BODY)
                (ECASE (FIRST SPEC)
                    (:FILES (IL:DOFILESLOAD (REST SPEC)))
                    ((:WINDOW :ROOM) )
                    (:VERSION
                       (SETQ VERSION (SECOND SPEC))
                       (WHEN (> VERSION *DEFSUITE-VERSION*)
                             (ERROR "DEFSUITE version ~S too high." VERSION)))))
            (UNLESS VERSION
                (WARN "No version found in DEFSUITE.  Presuming current.")
                (SETQ VERSION *DEFSUITE-VERSION*))

            ;; now make all the windows

            (UPDATE-PLACEMENTS)
            (DOLIST (SPEC SUITE-BODY)
                (CASE (FIRST SPEC)
                    (:WINDOW (LET ((WINDOW (RECONSTITUTE-WINDOW (FOURTH SPEC)
                                                  (NTHCDR 4 SPEC))))
                                  (WHEN (IL:WINDOWP WINDOW)
                                      (SETF (GETHASH (SECOND SPEC)
                                                   WINDOWS)
                                            WINDOW)
                                      (UNLESS (FIND-PLACEMENT WINDOW)
                                             (HIDE-WINDOW WINDOW)))))))

            ;; finally make the rooms

            (DOLIST (SPEC SUITE-BODY)
                (CASE (FIRST SPEC)
                    (:ROOM (APPLY #'INSTALL-ROOM WINDOWS (REST SPEC))))))))


(DEFUN INSTALL-ROOM (WINDOWS NAME &REST REST-KEYS &KEY PLACEMENTS &ALLOW-OTHER-KEYS)
   (APPLY 'MAKE-ROOM NAME :PLACEMENTS (WITH-COLLECTION (DOLIST (PLACEMENT-SPEC PLACEMENTS)
                                                          (LET ((PLACEMENT (APPLY #'INSTALL-PLACEMENT WINDOWS
                                                                                  PLACEMENT-SPEC)))
                                                               (WHEN PLACEMENT (COLLECT PLACEMENT)))))
          (LET ((REST (COPY-LIST REST-KEYS)))
               (REMF REST :PLACEMENTS)
               REST)))


(DEFUN INSTALL-PLACEMENT (WINDOWS NAME &REST REST-KEYS &KEY REGION SHRUNKEN? ICON-POSITION
                                    &ALLOW-OTHER-KEYS)
   (LET ((WINDOW (GETHASH NAME WINDOWS)))
        (WHEN WINDOW
            (MAKE-PLACEMENT-INTERNAL :WINDOW WINDOW :REGION (INTERNALIZE-REGION REGION)
                    :SHRUNKEN? SHRUNKEN? :ICON-POSITION (WHEN ICON-POSITION (INTERNALIZE-POSITION ICON-POSITION))
                    :PROPS
                    (LET ((PROPS (COPY-LIST REST-KEYS)))
                         (DOLIST (KEYWORD '(:REGION :SHRUNKEN? :ICON-POSITION))
                             (REMF PROPS KEYWORD))
                         PROPS)))))
```

```
(SEDIT:DEF-LIST-FORMAT DEFSUITE :ARGS (NIL :KEYWORD NIL)
   :INDENT (1))
```

;; deleting suites

```
(DEFUN DELETE-SUITE (SUITE-NAME &OPTIONAL ROOMS-TOO?)
   ;; delete all the rooms in the suite
   (WHEN ROOMS-TOO?
       (DOLIST (ROOM-NAME (SUITE-ROOMS SUITE-NAME))
           (LET ((ROOM (ROOM-NAMED ROOM-NAME)))
                (WHEN ROOM (DELETE-ROOM ROOM)))))
   ;; delete the suite
   (REMHASH SUITE-NAME *SUITES*))
```

;; constructing suites

```
(DEFUN DUMP-SUITE (SUITE-NAME &OPTIONAL QUIET?)
   (UPDATE-SUITE SUITE-NAME)
   ;; dump it to a file
   (LET ((FILE (OR (FIRST (IL:WHEREIS SUITE-NAME 'IL:SUITES))
                   (INTERN (NAMESTRING (MAKE-PATHNAME :NAME (STRING-UPCASE SUITE-NAME)
                                              :TYPE *SUITE-FILE-TYPE* :HOST NIL :DEVICE NIL :DIRECTORY NIL))
                          "IL")))
         FULL-NAME)
        (UNLESS (AND (IL:HASDEF FILE 'IL:FILES)
                     (IL:INFILECOMS? SUITE-NAME 'IL:SUITES (IL:FILECOMS FILE)))
            (IL:PUTDEF FILE 'FILE-ENVIRONMENTS `(DEFINE-FILE-ENVIRONMENT ,FILE :PACKAGE (DEFPACKAGE "ROOMS"
                                                                                           (:USE "LISP"
                                                                                                 "XCL")
                                                                                           (:SHADOW CL:ROOM)
                                                                                           )
                                                   :READTABLE "XCL"
                                                   :COMPILER :COMPILE-FILE))
            (IL:PUTDEF FILE 'IL:FILES `(((IL:FILES IL:ROOMS)
                                        (FILE-ENVIRONMENTS ,FILE)
                                        (IL:SUITES ,SUITE-NAME)))))
        (UNLESS QUIET? (NOTIFY-USER "Making file ~A ..." FILE))
        (IL:ALLOW.BUTTON.EVENTS)
        (SETQ FULL-NAME (PATHNAME (IL:MAKEFILE FILE '(IL:FAST IL:NEW))))
        (UNLESS QUIET?
            (NOTIFY-USER "Made file ~A" (NAMESTRING FULL-NAME)))
        (SETQ IL:NOTCOMPILEDFILES (REMOVE FILE IL:NOTCOMPILEDFILES))
        (SETQ IL:NOTLISTEDFILES (REMOVE FILE IL:NOTLISTEDFILES))
        FULL-NAME))
```

```
(DEFUN UPDATE-SUITE (SUITE-NAME)
   (LET ((IL:DFNFLG 'IL:PROP))
        (IL:PUTDEF SUITE-NAME 'IL:SUITES (CONSTRUCT-DEFSUITE SUITE-NAME (SUITE-ROOMS SUITE-NAME)))))
```

```
(DEFUN CONSTRUCT-DEFSUITE (SUITE-NAME ROOM-NAMES)
   (LET* ((ROOMS (WITH-COLLECTION (DOLIST (NAME ROOM-NAMES)
                                      (LET ((ROOM (ROOM-NAMED NAME)))
                                           (IF ROOM
                                               (COLLECT ROOM)
                                               (WARN "No room named ~S exists." NAME))))))
          (WINDOW-NAMES (MAKE-HASH-TABLE))
          (WINDOW-ABSTRACTIONS (CONSTRUCT-WINDOWS (ALL-WINDOWS-IN-ROOMS ROOMS)
                                      WINDOW-NAMES)))
         `(DEFSUITE ,SUITE-NAME
              (:VERSION ,*DEFSUITE-VERSION*)
              (:FILES ,@(CONSTRUCT-FILES WINDOW-NAMES))
              ,@WINDOW-ABSTRACTIONS
              ,@(MAPCAR #'(LAMBDA (ROOM)
                                `(:ROOM ,(ROOM-NAME ROOM)
                                        :PLACEMENTS
                                        ,(CONSTRUCT-PLACEMENTS ROOM WINDOW-NAMES)
                                        :INCLUSIONS
                                        ,(ROOM-INCLUSIONS ROOM)
                                        :BACKGROUND
                                        ,(BACKGROUND-EXTERNAL-FORM (ROOM-BACKGROUND ROOM))
                                        ,@(ROOM-PROPS ROOM)))
```

```
                        ROOMS))))
```

(DEFUN **ALL-WINDOWS-IN-ROOMS** (ROOMS)

;;; return a list containing all the windows in ROOMS

```
   (UPDATE-PLACEMENTS)
   (LET (WINDOWS)
        (DOLIST (ROOM ROOMS)
            (DOLIST (PLACEMENT (ROOM-PLACEMENTS ROOM))
                (PUSHNEW (PLACEMENT-WINDOW PLACEMENT)
                       WINDOWS :TEST 'EQ)))
        (NREVERSE WINDOWS)))
```

(DEFUN **CONSTRUCT-WINDOWS** (WINDOWS WINDOW-NAMES)

;;; construct the list of window abstractions for WINDOWS. store a name for each window in WINDOW-NAMES.

```
   (WITH-COLLECTION (LET ((WINDOW-NUMBER 0))
                         (DOLIST (WINDOW WINDOWS)
                             (LET ((ABSTRACTION (ABSTRACT-WINDOW WINDOW)))
                                  (WHEN ABSTRACTION
                                      (COLLECT `(:WINDOW ,WINDOW-NUMBER ,@ABSTRACTION))
                                      (SETF (GETHASH WINDOW WINDOW-NAMES)
                                            WINDOW-NUMBER)
                                      (INCF WINDOW-NUMBER))))))
```

(DEFUN **CONSTRUCT-FILES** (WINDOW-NAMES)

;;; returns the appended list of all the :FILES properties of all the window types of all the windows in WINDOW-NAMES.

```
   (LET ((ALL-FILES NIL))
        (MAPHASH #'(LAMBDA (WINDOW NAME)
                          (DOLIST (FILE (WINDOW-TYPE-PROP (WINDOW-TYPE WINDOW)
                                                :FILES))
                              (PUSHNEW FILE ALL-FILES)))
               WINDOW-NAMES)
        (REVERSE ALL-FILES)))
```

(DEFUN **CONSTRUCT-PLACEMENTS** (ROOM WINDOW-NAMES)

;; construct a list of external representations for the placements in ROOM, using the hash table WINDOW-NAMES to name windows. these external
;; representations are installed by INSTALL-PLACEMENT

```
   (WITH-COLLECTION (DOLIST (PLACEMENT (ROOM-PLACEMENTS ROOM))
                        (LET ((WINDOW-NAME (GETHASH (PLACEMENT-WINDOW PLACEMENT)
                                                   WINDOW-NAMES)))
                             (WHEN WINDOW-NAME
```

;; unnamed windows are ones that could not be abstracted -- we ignore them here.

```
                                 (COLLECT `(,WINDOW-NAME :REGION ,(EXTERNALIZE-REGION (PLACEMENT-REGION
                                                                                       PLACEMENT))
                                                  ,@(WHEN (PLACEMENT-ICON-POSITION PLACEMENT)
                                                          (LIST :SHRUNKEN? (PLACEMENT-SHRUNKEN? PLACEMENT)
                                                                :ICON-POSITION
                                                                (EXTERNALIZE-POSITION (PLACEMENT-ICON-POSITION
                                                                                       PLACEMENT))))
                                                  ,@(PLACEMENT-PROPS PLACEMENT)))))))
```

(DEFUN **ROOMS-NOT-IN-ANY-SUITE** (&OPTIONAL FOR-DELETION?)

;;; returns a list of all the rooms which are not in any suite

```
   (WITH-COLLECTION (DOLIST (ROOM (ALL-ROOMS T))
                        (LET ((NAME (ROOM-NAME ROOM)))
                             (UNLESS (OR (UNLESS FOR-DELETION?
```

;; Original & Pockets implicitly belong to the bootstrap suite & we don't want them added to
;; others.

```
                                             (OR (EQUAL NAME "Pockets")
                                                 (EQUAL NAME "Original")))
                                         (FIND-SUITE-CONTAINING NAME))
                                 (COLLECT ROOM)))))
```

(DEFUN **FIND-SUITE-CONTAINING** (ROOM-NAME)

```
   (MAPHASH #'(LAMBDA (SUITE-NAME ROOM-NAMES)
                     (WHEN (MEMBER ROOM-NAME ROOM-NAMES :TEST 'EQUAL)
                           (RETURN-FROM FIND-SUITE-CONTAINING SUITE-NAME)))
          *SUITES*))
```

(DEFUN **AUGMENT-SUITE** (SUITE-NAME ROOM-NAME)

```
   (PUSHNEW ROOM-NAME (SUITE-ROOMS SUITE-NAME)
         :TEST
         'EQUAL))


(DEFUN DELETE-ROOM-FROM-SUITE (ROOM-NAME SUITE-NAME)
   (SETF (SUITE-ROOMS SUITE-NAME)
         (DELETE ROOM-NAME (SUITE-ROOMS SUITE-NAME)
                :TEST
                'EQUAL)))

;; interactive code


(DEFGLOBALPARAMETER *SUITE-MENU-ITEMS*
   '(("Save Suite" '(WITH-BUTTON '(INTERACTIVE-DUMP-SUITE)
                          "Save Suite" "Save a set of rooms to a file")
           "Save a set of rooms to a file"
           (IL:SUBITEMS ("Update Suite" '(WITH-BUTTON '(INTERACTIVE-UPDATE-SUITE)
                                                 "Update Suite" "Update the DEFSUITE form of a suite")
                               "Update the DEFSUITE form of a suite")))
     ("Restore Suite" '(WITH-BUTTON '(INTERACTIVE-LOAD-SUITE)
                             "Restore Suite" "Load a set of rooms from a file")
           "Load a set of rooms from a file")
     ("Show Suite" '(WITH-BUTTON '(INTERACTIVE-SHOW-SUITE)
                          "Show Suite" "List the rooms in a suite")
           "List the rooms in a suite")
     ("Augment Suite" '(WITH-BUTTON '(INTERACTIVE-AUGMENT-SUITE)
                             "Augment Suite" "Add a room to a suite")
           "Add a room to a suite")
     ("Delete Suite" '(WITH-BUTTON '(INTERACTIVE-DELETE-SUITE)
                            "Delete Suite" "Delete a suite, and optionally all the rooms in it.")
           "Delete a suite, and optionally all the rooms in it."
           (IL:SUBITEMS ("Delete Room From Suite" '(WITH-BUTTON '(INTERACTIVE-DELETE-ROOM-FROM-SUITE)
                                                           "Delete Room From Suite"))))))


(DEFGLOBALVAR *SUITE-MENU* NIL)


(DEFUN SUITE-MENU ()
   (OR *SUITE-MENU* (SETQ *SUITE-MENU* (IL:CREATE IL:MENU
                                              IL:ITEMS IL:_ *SUITE-MENU-ITEMS*
                                              IL:TITLE IL:_ "Suites"
                                              IL:CENTERFLG IL:_ T)))
   (LET* ((ITEM (IL:MENU *SUITE-MENU*)))
         (WHEN ITEM                                      ; to be 100% compatible w/ background menu
             (IL:EVAL ITEM))))


(DEFUN INTERACTIVE-LOAD-SUITE ()
   (LET
    ((SUITE-NAME (PROMPT-FOR-SUITE-NAME)))
    (WHEN SUITE-NAME
        (IF (SUITE-ROOMS SUITE-NAME)
            (NOTIFY-USER "A suite named ~S already exists." SUITE-NAME)
            (LET
             ((FOUND (IL:FINDFILE (NAMESTRING (MERGE-PATHNAMES SUITE-NAME
                                                     (MAKE-PATHNAME :TYPE *SUITE-FILE-TYPE*

                                                            ;; override default defaults

                                                            :HOST NIL :DEVICE NIL :DIRECTORY NIL :VERSION NIL)
                                                     NIL))
                            T *SUITE-DIRECTORIES*)))
             (IF FOUND
                 (IL:ADD.PROCESS
                  `(FUNCALL ',#'(LAMBDA NIL (LET ((LOAD-COMPLETED? 'NIL))
                                                (UNWIND-PROTECT
                                                    (PROGN (LOAD FOUND)
                                                           (SETQ LOAD-COMPLETED? T))
                                                    (LET ((WINDOW (IL:WFROMDS NIL T)))
                                                         (WHEN WINDOW (IL:CLOSEW WINDOW)))
                                                    (WHEN LOAD-COMPLETED? (NOTIFY-USER "Restored suite ~S."
                                                                                 SUITE-NAME))))))
                  'IL:NAME "Restore Suite")
                 (NOTIFY-USER "Can't find suite ~S on *SUITE-DIRECTORIES*" SUITE-NAME)))))))


(DEFUN INTERACTIVE-DUMP-SUITE ()
   (LET ((SUITE-NAME (SELECT-SUITE :REASON "Save" :ALLOW-NEW? T)))
        (WHEN SUITE-NAME
            (WITH-BUTTON `(IL:RESETFORM (IL:TTYDISPLAYSTREAM (GET-MESSAGE-STREAM))
                                 (DUMP-SUITE ',SUITE-NAME))
                   (FORMAT NIL "Save ~A" SUITE-NAME)
                   (FORMAT NIL "Save suite named ~S." SUITE-NAME)))))
```

```
(DEFUN INTERACTIVE-MAKE-SUITE ()
   (LET ((ROOMS (ROOMS-NOT-IN-ANY-SUITE)))
        (IF (NULL ROOMS)
            (NOTIFY-USER "All rooms are already in some suite.")
            (LET ((SUITE-NAME (PROMPT-FOR-SUITE-NAME)))
                 (WHEN SUITE-NAME
                     (IF (SUITE-ROOMS SUITE-NAME)
                         (NOTIFY-USER "A suite named ~S already exists" SUITE-NAME)
                         (LET ((ROOM-NAMES (WITH-COLLECTION (NOTIFY-USER "Select rooms to go in ~S.~%Click
                                                                   outside menu when finished." SUITE-NAME)
                                                  (LOOP (WHEN (NULL ROOMS)
                                                              (RETURN))
                                                        (LET ((ROOM (SELECT-ROOM :FROM-ROOMS ROOMS)))
                                                             (COND
                                                                (ROOM (SETQ ROOMS (DELETE ROOM ROOMS :TEST
                                                                                                      'EQ))
                                                                      (COLLECT (ROOM-NAME ROOM)))
                                                                (T (RETURN))))))))
                              (WHEN ROOM-NAMES (MAKE-SUITE SUITE-NAME ROOM-NAMES)))))))))


(DEFUN INTERACTIVE-UPDATE-SUITE ()
   (LET ((SUITE-NAME (SELECT-SUITE :REASON "Update" :ALLOW-NEW? T)))
        (WHEN SUITE-NAME
            (WITH-BUTTON `(IL:RESETFORM (IL:TTYDISPLAYSTREAM (GET-MESSAGE-STREAM))
                                 (UPDATE-SUITE ',SUITE-NAME))
                   (FORMAT NIL "Update ~A" SUITE-NAME)
                   (FORMAT NIL "Update suite named ~S." SUITE-NAME)))))


(DEFUN INTERACTIVE-DELETE-SUITE ()
   (LET ((SUITE-NAME (SELECT-SUITE :REASON "Delete")))
        (WHEN (AND SUITE-NAME (CONFIRM "Delete suite ~S?" SUITE-NAME))
            (DELETE-SUITE SUITE-NAME (CONFIRM "Delete all rooms in ~S too? (will close windows)" SUITE-NAME))
            (NOTIFY-USER "Suite ~S deleted." SUITE-NAME))))


(DEFUN INTERACTIVE-SHOW-SUITE ()
   (LET ((SUITE-NAME (SELECT-SUITE :REASON "Show")))
        (WHEN SUITE-NAME
            (NOTIFY-USER "Suite ~S contains rooms:~{ ~S~}." SUITE-NAME (SUITE-ROOMS SUITE-NAME)))))


(DEFUN INTERACTIVE-AUGMENT-SUITE ()
   (LET ((ROOMS (ROOMS-NOT-IN-ANY-SUITE)))
        (IF (NULL ROOMS)
            (NOTIFY-USER "All rooms are already in some suite.")
            (LET ((SUITE-NAME (SELECT-SUITE :REASON "Augment Suite")))
                 (WHEN SUITE-NAME
                     (NOTIFY-USER "Select room to add to suite ~S" SUITE-NAME)
                     (LET ((ROOM-NAME (SELECT-ROOM :FROM-ROOMS ROOMS :NAME-ONLY? T)))
                          (WHEN ROOM-NAME (AUGMENT-SUITE SUITE-NAME ROOM-NAME))))))))


(DEFUN INTERACTIVE-DELETE-ROOM-FROM-SUITE ()
   (LET ((SUITE-NAME (SELECT-SUITE :REASON "Delete room from")))
        (WHEN SUITE-NAME
            (LET ((ROOM-NAME (MENU (SUITE-ROOMS SUITE-NAME)
                                   "Select Room")))
                 (WHEN ROOM-NAME
                     (DELETE-ROOM-FROM-SUITE ROOM-NAME SUITE-NAME)
                     (NOTIFY-USER "Deleted room ~S from suite ~S." ROOM-NAME SUITE-NAME))))))


(DEFUN SELECT-SUITE (&KEY REASON ALLOW-NEW?)
   (LET ((ITEMS (WHEN ALLOW-NEW?
                    '(("<new suite>" :NEW)))))
        (MAPHASH #'(LAMBDA (SUITE-NAME SUITE-BODY)
                          (PUSH SUITE-NAME ITEMS))
               *SUITES*)
        (IF ITEMS
            (LET ((SUITE-NAME (MENU ITEMS REASON)))
                 (IF (AND ALLOW-NEW? (EQ SUITE-NAME :NEW))
                     (INTERACTIVE-MAKE-SUITE)
                     SUITE-NAME))
            (PROGN (NOTIFY-USER "No suites!")
                   NIL))))


(DEFUN PROMPT-FOR-SUITE-NAME (&OPTIONAL (PROMPT "Suite Name:"))
   (LET ((STRING (PROMPT-USER PROMPT)))
        (WHEN STRING (STRING-UPCASE STRING))))

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

{MEDLEY}<rooms>ROOMS-SUITES.;1   28-Jun-2024 18:34:03
-- Listed on 30-Jun-2024 13:15:15 --

FUNCTION INDEX

VARIABLE INDEX

SEDIT-FORMAT INDEX

MACRO INDEX

FILE-ENVIRONMENT INDEX

DEFINER INDEX

CONSTANT INDEX

DEFINE-TYPE INDEX