Envos

ROOMS

ROOMS™

en·vōs

300001
Medley Release
September 1988

Address comments to:
Envos Corporation
1157 San Antonio Road
Mountain View, CA 94043

ROOMS™

300001

Medley Release

September, 1988

Text was written and produced on Xerox Artificial Intelligence workstations using Xerox printers to produce text masters. The typeface is Optima.

TABLE of CONTENTS

LIST of FIGURES

[This page intentionally left blank]

LIST of TABLES

[This page intentionally left blank]

PREFACE

Welcome to ROOMS, a sophisticated way to handle the problem of a cluttered screen. ROOMS provides you with a new way to manage your Envos Software Development Environment by allowing you to create the equivalent of additional screens to group windows used for a common task.

The purpose of this manual is to give you a comprehensive guide to using ROOMS at both the menu level and programmatically. This manual includes a Programmer's Guide that lets you take the cover off a little (well tilt it up anyway). You don't have to use the Programmer's Guide or even read it if you are content to use ROOMS at the menu level. ROOMS provides a wide range of functionality without using the Programmer's Guide.

What's In the Manual

Chapter by chapter here's what you'll find in the ROOMS manual.

- Chapter 1, Introduction, gives you a brief overview of ROOMS and some of its features.
- Chapter 2, Loading ROOMS Software, tells you how to load the ROOMS software (from both floppy and tar tape), what fonts are required and tells you how to load the ROOMS on-line tutorial, the ROOMS Introduction.
- Chapter 3, Rooms, shows you how to create the virtual workspaces called rooms and demonstrates how to customize a room's appearance using the Lisp structure editor SEdit.
- Chapter 4, Placements, explains how ROOMS handles the capability for displaying a window in more than one room. Placements enhance the standard window system and this chapter explains how that is done and how you can use it.
- Chapter 5, Navigation, shows you the various ways that ROOMS provides for moving between rooms. Also included is information on buttons.
- Chapter 6, The Overview, explains how to use the ROOMS Overview, a special room that allows you see all your rooms (and what's in them) at once. Also included is information on how to edit the placements of a room outside the Overview.
- Chapter 7, Suites, explains the ROOMS suite interface. Suites allow you to save sets of rooms on files and reload the suite files thus allowing you to easily restore a past state.
- Appendix A, Programmer's Guide to ROOMS, is a comprehensive reference to the programmatic interface to rooms. Many of the things that you do in rooms using menus you can duplicate using Lisp functions.

At the end of the manual there is a glossary of ROOMS terms and an index.

How to Use this Manual

For first time users of ROOMS we recommend that you load and use the ROOMS Introduction, an interactive tutorial that takes you on a tour of ROOMS and introduces ROOMS features by teaching you to use them.

Once you've gone through the tutorial you're ready to use ROOMS to make your life easier. Use the manual as a reference, referring to it when you need some instructions on how to do something particularly vexing and not clear. Hopefully, we've anticipated your confusion and you will find the answers to your questions easily accessible.

Once you feel you've exhausted the possibilites of using what ROOMS gives you, dive into the Programmer's Guide for some serious customizing. The Programmer's guide assumes you are familiar with the Common LISP style of programming as presented in *Common LISP the Language* by Guy Steele.

Finally, enjoy yourself, ROOMS is so much fun to use you may forget how much more efficient it makes you.

References

Lisp Release Notes, Medley Release, Appendix B, SEdit

Common LISP the Language by Guy Steele

Interlisp-D Reference Manual, Volumes 1-3, Koto release

Medley 1.0-S User's Guide

Acknowledgement

As with any other project of this type ROOMS is a collaborative effort with many people contributing their skills to its completion. First came Austin Henderson and Stu Card, pioneers in user interface research at Xerox's Palo Alto Research Center (Xerox PARC). Next came Doug Cutting a developer at Envos (formerly Xerox Artificial Intelligence Systems) who reworked much of the code to be more "Common LISPy." Mary Eward of Envos took the early code listings and wrote the first draft of the documentation. She handed her draft to Bob Weisblatt who wrote the manual you are now reading. Acknowledgement is also long overdue to Ron Clarke, the ROOMS project manager,

who steered the project through the shoals of development, documentation, testing, and release. Also, thanks are owed to Larry Harada for testing ROOMS and to Kat Kohlsaat who who helped test ROOMS and handled the release of the product. We on the ROOMS team are proud of the product we have developed and hope that you have as much fun using ROOMS as we had bringing it to you.

[This page intentionally left blank]

1. INTRODUCTION

What Is ROOMS

ROOMS effectively increases the size of your display by allowing you to create a set of virtual workspaces called rooms. Each room is equivalent to having another screen; you can create a new, empty room whenever you need more screen space.

Moreover, while each room you create can contain a unique set of windows, the environment is the same for all rooms. The number of rooms and the number of windows, either in a room or in the entire ROOMS system, is limited only by system constraints, e.g., VMem size.

ROOMS helps minimize the time you spend arranging windows. It allows you to create rooms that group windows used for a common task and to segregate one group of windows from another. To allow easy movement between rooms you create doors that lead from one room to another.

ROOMS also allows you to "paint the walls" of each room with a different background. This further enhances the distinctiveness of each room and thus its connection with a specific task.

ROOMS allows you to group rooms into suites and then save these suites to a file. You can store the suite files on your local disk, file server, or floppies and load them as you load any other Lisp file.

A Brief Explanation of Some ROOMS Features

Backgrounds

A background is the appearance of the screen behind all the windows. If you give each room a distinct background the time you spend orienting yourself, as you travel from room to room, decreases.

ROOMS software includes the background bitmaps, TILE-BITMAP, RENAISSANCE-BITMAP, and SQUARES-BITMAP (Figure 1-1). In addition you can create your own set of backgrounds. Instructions on how to install backgrounds are in Chapter 3, Rooms, "Edit a Room, :BACKGROUND."

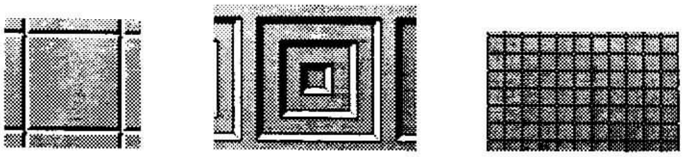


Figure 1-1. ROOMS background bitmaps. Left to right: TILE-BITMAP, RENAISSANCE-BITMAP, and SQUARES-BITMAP.

Inclusions

At times you may have windows that you want to include in some rooms but not in others. With Inclusions, you can set up a single room, called a "source" room that contains those windows. You then include the source room in several other rooms, called "destination" rooms. All the windows you put into the source room appear in the destination rooms.

For information on how to include one room in another, see Chapter 3, Rooms, "Edit A Room, :INCLUSIONS."

Suites

ROOMS provides you with a suite facility that allows you to:

- Group a set of rooms together into a suite.
- Save the suite to a file.
- Load the suite file, reestablishing a set of rooms without having to recreate each room individually.

Pockets

Pockets, a room automatically created when you first load ROOMS, lets you have a set of windows appear in every room. Pockets initially contains only the prompt window. Any window you put in Pockets automatically appears in every room in the system.

In general it is recommended that you use Inclusions rather than Pockets to handle your need for a standard set of windows being present in any room. This recommendation is based on the fact that you cannot add Pockets to a suite. However if you wish you can delete Pockets from the system in the Overview (see below).

Original

The Original room is automatically created when you load ROOMS. If Original and Pockets are the only rooms present then all the windows that normally appear on your screen are in Original. You can delete Original from the system in the Overview.

The Overview

The Overview is a special room that allows you to see all existing rooms at the same time. While in the Overview you can manipulate rooms and windows using keyboard-mouse combinations. A special feature of the Overview is that you can delete the two rooms that ROOMS creates when it is first loaded—Pockets and Original.

Conventions

Throughout this manual, you there are references to ROOMS and to Rooms (or rooms). ROOMS in all uppercase letters refers to the application itself; a word with lower case letters (Rooms or rooms) refers to specific rooms within the ROOMS system.

Use of Upper and Lower Case in Code Examples

This manual uses two styles for showing code examples depending on the context of the example itself. In the first seven chapters the examples generally are illustrations that use the Lisp structure editor, SEdit. Since SEdit displays typein in all uppercase by default, the examples are given in all uppercase.

However, in Appendix A examples are illustrated using the "Common Lisp" style, i.e., all lower case.

Keyboard Conventions

Keys that you press are in uppercase (e.g., COPY, for the Copy key). A carriage return is displayed as <RETURN>.

There are some keys and key combinations that have the same effect as others (e.g., SHIFT has the same effect as COPY when used analogously). Throughout this manual most differences and alternatives have been ignored but they still work; that is you can still use SHIFT for COPY, CONTROL-SHIFT for MOVE, etc.

[This page intentionally left blank]

2. LOADING ROOMS SOFTWARE

When you load ROOMS, you can either load only the software or the software plus a ROOMS Introduction suite. You load ROOMS and the Introduction from either floppies or a tar tape depending on the hardware you are using. This chapter has instructions for loading from both floppies and a tar tape.

Loading the Software

Loading ROOMS from Floppy

This procedure loads only the ROOMS software.

1. Set `IL:DISPLAYFONTDIRECTORIES` to the location of the font HELVETICA18-MRR-C0.DISPLAYFONT.
2. With the ROOMS floppy disk in the disk drive, type to an Exec:

```
DIR {FLOPPY}
```

3. After the floppy directory prints type in an Exec:

```
(LOAD "{FLOPPY}ROOMS.DFASL")
```

   - Loading ROOMS.DFASL automatically loads all the other necessary files.

When the load is finished, you have an initial set-up containing two rooms: a room called Original that contains whatever would normally be on your screen and a room called Pockets. The purpose of the Pockets room is described in Chapter 1, Introduction, "Pockets."

Most of what you do with ROOMS is controlled from the ROOMS submenu (Figure 2-1). Each menu item is explained in this manual.

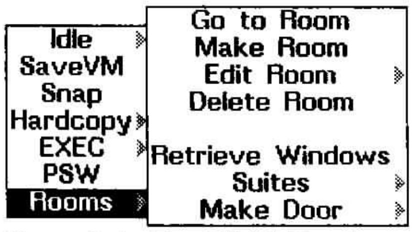


Figure 2-1. The ROOMS submenu

Loading ROOMS and the Introduction Suite from Floppy

This procedure loads both the ROOMS software and the ROOMS Introduction. The Introduction introduces you to basic ROOMS concepts through illustrations and exercises. We encourage new users to load this tutorial.

Before loading the Introduction, you must make the following font files accessible to ROOMS. These font files are supplied with the Envos Lisp Software kit:

HELVETICA12-MRR-C0.DISPLAYFONT
HELVETICA14-MRR-C0.DISPLAYFONT
HELVETICA18-MRR-C0.DISPLAYFONT
HELVETICAD24-MRR-C0.DISPLAYFONT
MODERN24-MRR-C0.DISPLAYFONT

ROOMS uses the variable `IL:DISPLAYFONTDIRECTORIES` to determine where to look for the fonts it needs.

1. To load the ROOMS Introduction type to an Exec:

   ```
   DIR {FLOPPY}
   ```

2. After the floppy directory prints in the Exec type:

   ```
   (LOAD "{FLOPPY}ROOMS-INTRO.DFASL")
   ```

The ROOMS Introduction contains nine rooms: a room called Original that contains your regular screen image (with one change), a room called Pockets, and seven rooms containing the ROOMS Introduction. The Original room has a window placed in the middle of the screen with the words "Enter Introduction" written on it. Select this window with the left mouse button to start the tutorial.

When you are finished with the ROOMS Introduction suite, you can delete the suite from your environment with the "Delete Suite" command (see Chapter 7, Suites).

Loading ROOMS and/or the ROOMS Introduction into Medley 1.0-S

The procedure in this section assumes that you have ROOMS on a tar tape, that you need to copy the software from the tape so you can load ROOMS into Medley 1.0-S, that you are familiar with using Medley 1.0-S, and UNIX. You cannot load ROOMS directly from the tape.

The following instructions show you how to copy the software to your home directory and then load ROOMS into Medley 1.0-S.

1. Login to the C-Shell
   - You cannot be logged in to more than one workstation on the net.
2. Insert the tape into the drive.
3. Type to the C-Shell:

   prompt% `cd`

   and press <RETURN>.

   - Connects you to your home directory.
4. Copy the ROOMS software from the tape. Indicate the appropriate device abbreviation for your tape by replacing XX in the example below with:

   `ar` for the Archive drive

   `st` for a SCSI tape drive

   `mt` for the Tapemaster half-inch (1600 bpi) drive.

   The command entry sequence is:

prompt% `tar xvf /dev/rXX0`

and press <RETURN>.

- Copies the contents of the tape to the connected directory. The tape has the ROOMS software, the ROOMS Introduction and the unsupported ROOMS Users' Modules on it (You can load the ROOMS Users' modules from "{DSK}rooms/users/".)

4. To load ROOMS into Medley 1.0-S type in an Exec:

```
(LOAD "{DSK}rooms/system/rooms.dfasl")
```

5. To load the ROOMS Introduction type in an Exec:

```
(LOAD "{DSK}rooms/system/rooms-intro.dfasl")
```

- This loads ROOMS also if it's not already loaded.

Note: See the section "Loading ROOMS and the Introduction Suite from Floppy" above for a list of the fonts that ROOMS must have access to when you load the Introduction.

Adding the ROOMS Introduction with ROOMS Loaded

If you have already loaded the ROOMS software and would like to add the ROOMS Introduction suite, you simply have to load ROOMS-INTRO.DFASL. The Introduction suite is added to the rooms already present and the "Enter Introduction" window appears in the room you are in when you load the Introduction.

[This page intentionally left blank]

3. ROOMS

A room is a virtual workspace that you create and design. As such, it normally displays only those windows necessary for the task associated with that workspace (though you are not limited to the windows present in any room). Therefore, you move from room to room in order to move from one task to another. Descriptions of how to move around are given in Chapter 5, Navigation.

This chapter covers:

- How to create a room.
- How to delete a room.
- How to change a room's appearance.
- How to include one or more rooms in other rooms.

You can also manipulate rooms from the Overview, a representation of all rooms in the environment. For an explanation of the Overview see Chapter 6, The Overview.

Note: When you are designing rooms and deciding what windows to put into them you should include a prompt window in each room. ROOMS makes extensive use of the prompt window to print messages about what is going on. Also, if you go into Idle from a room that doesn't have a prompt window when you attempt to exit Idle there is no prompt window visible to enter your name and password. You can still login: just type your name and password. But you won't be able to see what's being printed in the prompt window because it's in another room.

Manipulating Rooms

"Make Room"—Creating a Room

To create a new room select "Make Room" on the ROOMS submenu (Figure 3-1).

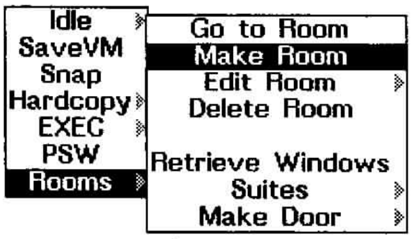


Figure 3-1. "Make Room" menu option

ROOMS asks you for the name of the new room in the prompt window. Type in the name and press <RETURN>; ROOMS adds the room to the current set. You are not moved to that room.

Initially, a new room contains only those windows that are in Pockets. If you've deleted Pockets then a new room contains nothing and has the default light gray background.

"Delete Room"

Deleting a room removes it from the current set of rooms.

To delete a room select "Delete Room" on the ROOMS submenu (Figure 3-2).

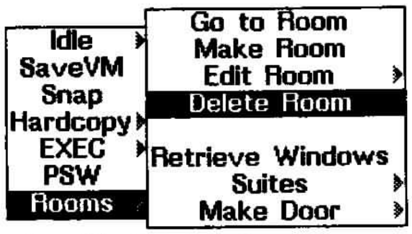


Figure 3-2. The "Delete Room" menu option

ROOMS responds with a menu of all the existing rooms that are *not* part of a suite (Figure 3-3). After selecting the room to delete press the left mouse button to confirm the deletion. Pressing the middle or right mouse button aborts the deletion.

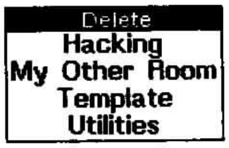


Figure 3-3. The "Delete" submenu

Note: Deleting a room also deletes the placements that are in the room.

See Chapter 6, Suites for information on deleting a room that is part of a suite.

"Edit Room"—Changing a Room's Appearance

Each room has a keyword list associated with it that contains information on the room's appearance. In order to change the appearance of a room you edit this keyword list using SEdit the Envos Lisp structure editor (see the *Release Notes*, Appendix B, SEdit). You do not have to be in a room to edit it.

You can add your own properties to a room using SEdit. You can also add a property programmatically, see Appendix A, The Programmer's Guide to ROOMS "Rooms" (the function `room-prop`).

To open an SEdit window with a room's description first select "Edit Room" from the ROOMS submenu (Figure 3-4).

Note: The "Edit This Room" command allows you to edit the room you're in (called the current room). "Exclude Room" and "Include Room" are explained below in ":Inclusions." "Edit Placements" is explained in Chapter 6, The Overview.

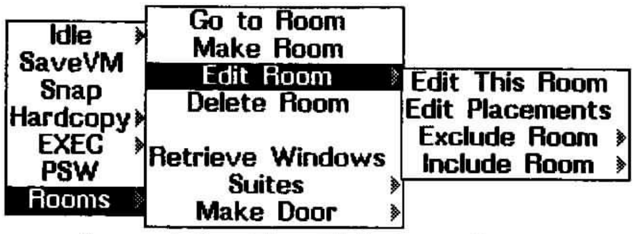


Figure 3-4. The "Edit Room" menu option

When you select "Edit Room," a menu opens listing all the existing rooms including those rooms contained in suites (Figure 3-5). Selecting the room you what to edit from the "Edit" menu (using any mouse button) opens an SEdit window with that room's description (Figure 3-6).

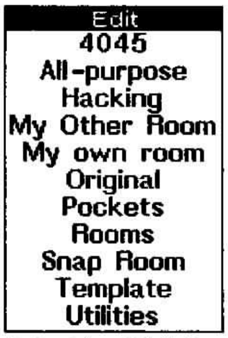
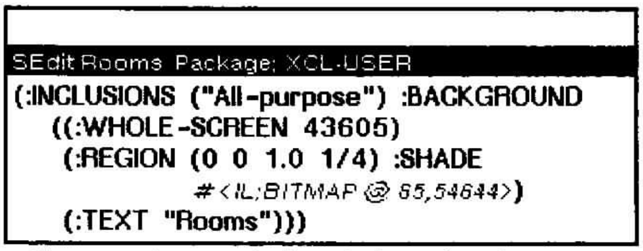


Figure 3-5. The "Edit" submenu

When you change a room's description, its appearance changes as soon as you end the edit session.



SEdit Rooms Package: XCL-USER

```
(:INCLUSIONS ("All-purpose") :BACKGROUND
   ((:WHOLE-SCREEN 43605)
    (:REGION (0 0 1.0 1/4) :SHADE
              #<IL:BITMAP @ 65,54644>)
    (:TEXT "Rooms")))
```

Figure 3-6. An SEdit window for the room Rooms

:INCLUSIONS

`:INCLUSIONS` allow you to include one or more rooms (called source rooms) and all their windows in other rooms (called destination rooms).

Inclusions are useful when you have a set of windows (e.g., an Exec, a mail watcher, the Who-line, etc.) that you want to appear in many, but not all, rooms. Thus inclusions are distinguished from Pockets in that any window present in Pockets appears in *every* room.

In addition, a permanent, dynamic link is set up between the source and destination rooms: when the source room is modified all destination rooms that include it are automatically modified. Editing a destination room has no effect on a source room. However, if you close or move a window that is actually from the source room the change is propagated back to the source room and then to any other rooms that include the source room. Copying or moving an included window first moves the placement for that window into the current room before executing the action.

How to Use Inclusions

Let's say you create two rooms, one called Template and the other Hacking. Template contains an Exec, a mail watcher, the Who-line and a prompt window. Hacking contains nothing. If you include Template in Hacking then all the windows in Template also appear in Hacking.

But Template's name is also included in Hacking. So in the lower left corner where you probably want just Hacking you have Hacking written over Template (Figure 3-7).

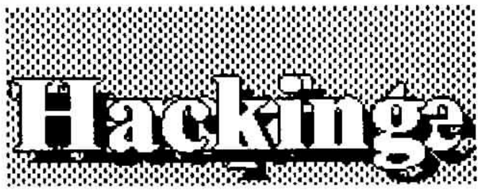
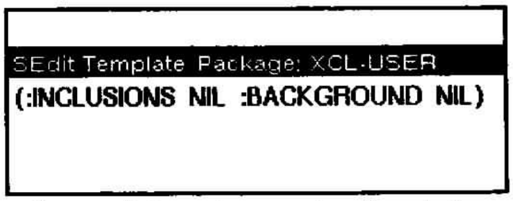


Figure 3-7. Two room names written one on top of another

To correct the problem you need to remove the text from the background description of Template (Figure 3-8).



SEdit Template Package: XCL-USER

```
(:INCLUSIONS NIL :BACKGROUND NIL)
```

Figure 3-8. A room's description without any text in it

If you move or close one of the three included windows in Hacking the action is duplicated in Template (and vice versa). If you want one (or more) of the included windows to be in different positions in each room you can copy windows into Hacking (see Chapter 4, Placements.)

You can also include backgrounds in rooms. See the :BACKGROUND section below for information on how to "paint" your rooms with different "colors" or "paper the walls" with "patterns."

"Include Room"

You can include rooms by selecting "Include Room" from the ROOMS submenu (Figure 3-9) (Selecting "In this Room" allows you to include rooms in the current room.)

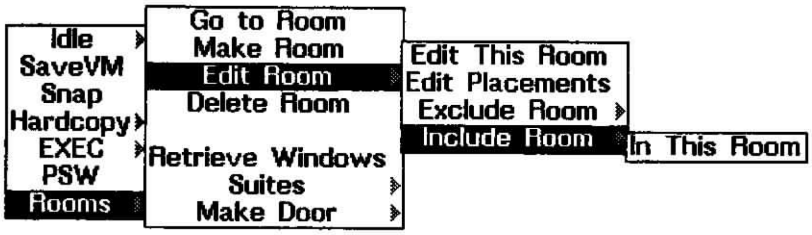


Figure 3-9. The "Include Room" menu option

When you release the right mouse button after selecting "Include Room" the "Include in . . ." menu opens. This menu lists all the rooms in the environment (Figure 3-10).

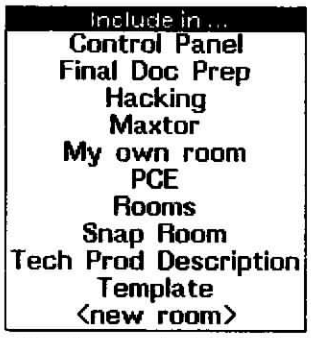


Figure 3-10. A sample "Include in" menu

The room that you choose (for illustrative purposes "Hacking" was chosen) in the "Include in" menu becomes the *destination* room (if you select <new room> the new room becomes the destination room); ROOMS deletes it from the menu and reopens the menu with the destination room as part of the menu's title (Figure 3-11).

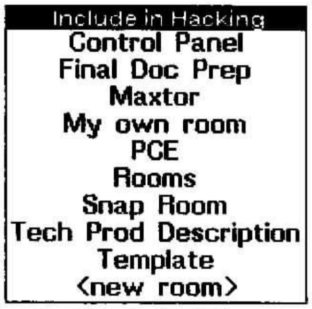
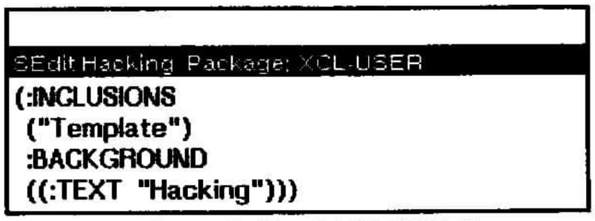


Figure 3-11. The "Include in" menu with a room name as part of the menu's title

The room you select from this menu is the *source* room; it is included in the first room you chose.

You can also include one room in another using SEdit to edit the room description. Figure 3-12 shows the SEdit window of the room Hacking with Template included.



```
SEdit Hacking Package: XCL-USER
(:INCLUSIONS
 ("Template")
 :BACKGROUND
 ((:TEXT "Hacking")))
```

Figure 3-12. :INCLUSIONS

Inclusions are inherited. For example if the room Tech Prod Description is included in Template, then Hacking would also include the room Tech Prod Description. For a complete explanation of how ROOMS handles inherited inclusions please consult Appendix A, Programmer's Guide to ROOMS, "Inclusions."

"Exclude Room"—Removing Inclusions

You can also use the ROOMS submenu to remove source rooms from destination rooms. First select "Exclude Room" from the Edit Room submenu (Figure 3-13). Selecting "In This Room" allows you to exclude rooms from the current room.

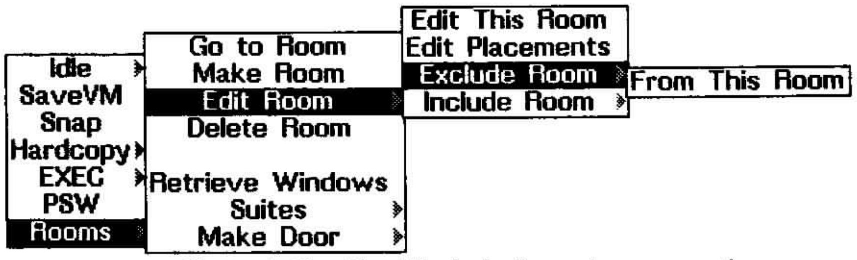


Figure 3-13. The "Exclude Room" menu option

When you release the right mouse button the "Exclude from" menu opens (Figure 3-14).

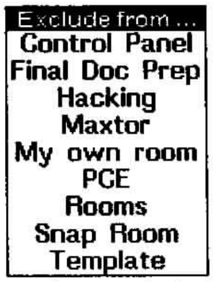


Figure 3-14. The "Exclude from" menu

Pick the room you want to exclude a room *from*, that is you are picking the destination room from this first menu. Remember, rooms are destination rooms if some other room is included in them. For illustrative purposes the room Hacking was chosen from this menu.

When you've picked the destination room, the name of that room is added to the menu's title and a menu opens listing all of the rooms that are included in the room you picked (Figure 3-15).

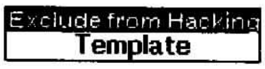


Figure 3-15. The "Exclude from" menu with a room name as part of the menu's title

Select the room you want to exclude. ROOMS deletes it from the :INCLUSIONS list. Any windows that were in the destination room only by virtue of being in the source room are removed. However they remain in the source room.

:BACKGROUND

Use :BACKGROUND to specify the look of the background in a room. Its value is a series of sublists, where each sublist defines a unique part of the background. Some examples of the values of `:BACKGROUND` are given below.

Note: Because of the way that ROOMS handles backgrounds you should not use `IL:CHANGEBACKGROUND` when specifying how you want a background to look.

`:EVAL` any time you need to do an evaluation in a background specification use `:EVAL`. For example, `(... :BACKGROUND (:EVAL IL:MY-BACKGROUND)...)` evaluates the variable `IL:MY-BACKGROUND` to find the background bitmap to use in that room. `:EVAL` is used only within the `:BACKGROUND` description. You can put any expression you want evaluated after `:EVAL`.

`:WHOLE-SCREEN` follow this keyword with a bitmap, shade or texture in order to change the background of the room. If you use a variable whose value is a bitmap, shade or texture, then you must use `:EVAL`.

Examples:

```
(:WHOLE-SCREEN 43605)
```

Changes the background to the shade given by 43605 (IL:GRAYSHADE).

```
(:WHOLE-SCREEN (:EVAL IL:MYBITMAP))
```

Tiles the background with the bitmap that is the value of the variable IL:MYBITMAP.

You can put a border around the entire background using `:BORDER` and specify a border shade using `:BORDER-SHADE`.

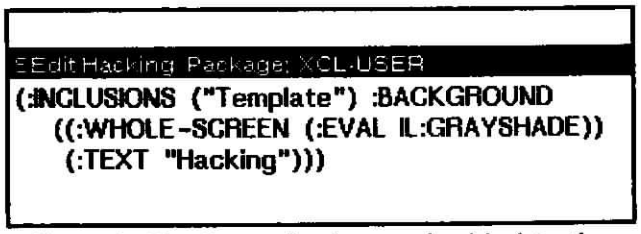
`:BORDER` in pixels; defaults to zero (0).

`:BORDER-SHADE` a shade; defaults to `IL:BLACKSHADE`.

```
(:WHOLE-SCREEN (:EVAL IL:MYBITMAP) :BORDER 4
     :BORDER-SHADE (:EVAL IL:GRAYSHADE))
```

Tiles the background with the bitmap that is the value of the variable IL:MYBITMAP and puts a gray border four pixels wide around the screen.

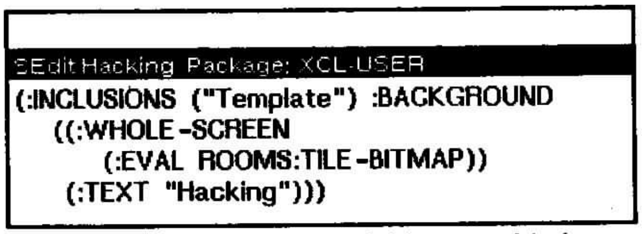
Thus, if you want to give the room Hacking (see above) a new background you would change Hacking's description (Figure 3-16).



```
SEdit Hacking Package: XCL-USER
(:INCLUSIONS ("Template") :BACKGROUND
   ((:WHOLE-SCREEN (:EVAL IL:GRAYSHADE))
    (:TEXT "Hacking")))
```

Figure 3-16. A new background added to the room named Hacking

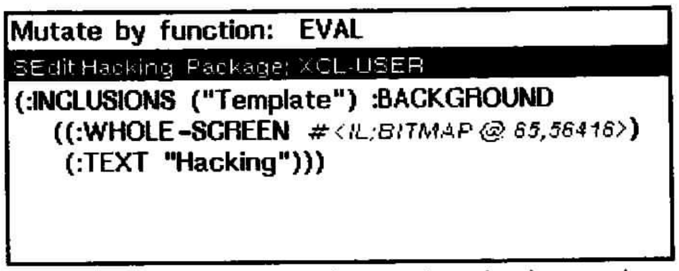
Instead of just a plain gray background you may want something a little more exciting. Hacking can be pretty messy so let's put tile up in the room! One of the bitmaps that comes with ROOMS is called TILE-BITMAP. To change the background of Hacking from gray to tiles you need to change Hacking's description again (Figure 3-17).



```
SEdit Hacking Package: XCL-USER
(:INCLUSIONS ("Template") :BACKGROUND
   ((:WHOLE-SCREEN
       (:EVAL ROOMS:TILE-BITMAP))
    (:TEXT "Hacking")))
```

Figure 3-17. A background bitmap added to the room Hacking

Note: If you used your own bitmap for the background of Hacking and you wanted to save the room as part of a suite (see Chapter 6, Suites) you would need to ensure that the bitmap was available when you loaded ROOMS or put the actual bitmap into the background description by mutating (Meta-Z) your bitmap by **eval** (Figure 3-18). For an explanation of how to use SEdit's mutate feature see the *Lisp Release Notes*, Appendix B, SEdit



```
Mutate by function: EVAL
SEdit Hacking Package: XCL-USER
(:INCLUSIONS ("Template") :BACKGROUND
   ((:WHOLE-SCREEN #<IL:BITMAP @ 65,56416>)
    (:TEXT "Hacking")))
```

Figure 3-18. Saving a bitmap in a background description

:REGION

:REGION lets you paint different regions of the screen with different shades, bitmaps, or textures. You can also specify each region to have different size and shaded borders. **:REGION** can have four arguments: a list specifying the region; a shade, bitmap or texture to fill that region of the screen (**:SHADE**); a border (:**BORDER**); and a border shade (**:BORDER-SHADE**).

Note: If you specify a region and give it a shade you must also specify a value for :WHOLE-SCREEN (see above). ROOMS first paints the entire background with the shade you give to :WHOLE-SCREEN and then paints the separately defined region.

Specifying a Region

The list giving the dimensions of the region should be in the form (LEFT BOTTOM WIDTH HEIGHT), where LEFT and BOTTOM specify the position of the lower left-hand corner of the region and WIDTH and HEIGHT specify the size of the region.

You can use integers, fractions or floating point numbers to specify each part of the region list. In fact, you can mix different types of numbers together in any region's specification. You can also specify more than one region in a room's description (Figure 3-20).

For example:

```
(:REGION (0 3/4 1.0 1/4) . . .)
```

Defines as a region the top quarter of the screen.

```
(:REGION (1/4 1/4 1/2 1/2) . . .)
```

Defines the region as a square centered in the middle of the screen.

In general the different types of numbers you use to specify the region's dimensions do the following:

INTEGER	specifies size or position in pixels
FRACTION or FLOATING POINT	specifies size or position in proportion to entire screen (1.0 is the entire screen).

Note: All positions can be specified proportionally.

Using IL:GETREGION. One way to specify regions, especially small ones, of the screen is by using `IL:GETREGION`. That is, include in your :REGION specification something like:

```
(:REGION (IL:GETREGION). . .)
```

Select `IL:GETREGION` (including the parentheses) and then mutate it by `eval`. You are asked to sweep out a region of the screen. The dimensions of that region are automatically placed in the `:REGION` specification in place of `IL:GETREGION` (see the *Interlisp-D Reference Manual*, Chapter 28 Windows and Menus, "Section 28.3 Interactive Display Functions" for information on using `IL:GETREGION`.)

:SHADE—"Painting"or "Papering" a Region of the Screen

`:SHADE` works similarly to `:WHOLE-SCREEN` but is used to paint regions smaller than the entire screen. You supply a shade, bitmap, or texture. If you want to use a variable use `:EVAL`.

For example:

`(:REGION (0 3/4 1.0 1/4) :SHADE (:EVAL IL:MYBITMAP))`

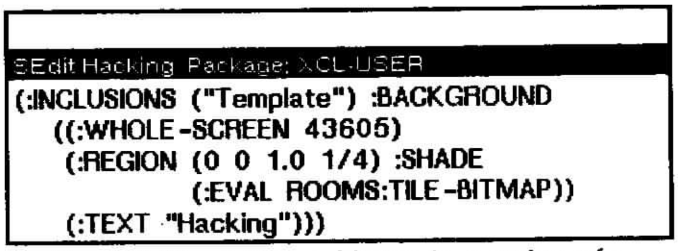
Tiles the top quarter of the screen with `IL:MYBITMAP`.

And,

`(:REGION (1/4 1/4 1/2 1/2) :SHADE 65535)`

Fills a square in the middle of the screen with the shade 65535 (`IL:BLACKSHADE`).

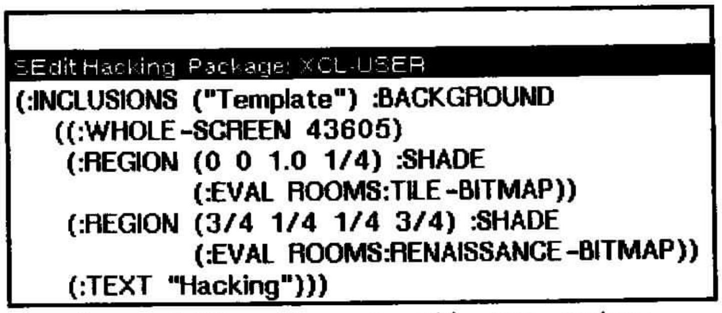
Now you can make the room Hacking even fancier. First let's make TILE-BITMAP fill just the bottom quarter of the screen (Figure 3-19).



```
(:INCLUSIONS ("Template") :BACKGROUND
   ((:WHOLE-SCREEN 43605)
    (:REGION (0 0 1.0 1/4) :SHADE
              (:EVAL ROOMS:TILE-BITMAP))
    (:TEXT "Hacking")))
```

Figure 3-19. Putting a bitmap in a region of a room

Let's keep going and put RENAISSANCE-BITMAP up the right quarter of the screen, but above the tile (Figure 3-20).



```
(:INCLUSIONS ("Template") :BACKGROUND
   ((:WHOLE-SCREEN 43605)
    (:REGION (0 0 1.0 1/4) :SHADE
              (:EVAL ROOMS:TILE-BITMAP))
    (:REGION (3/4 1/4 1/4 3/4) :SHADE
              (:EVAL ROOMS:RENAISSANCE-BITMAP))
    (:TEXT "Hacking")))
```

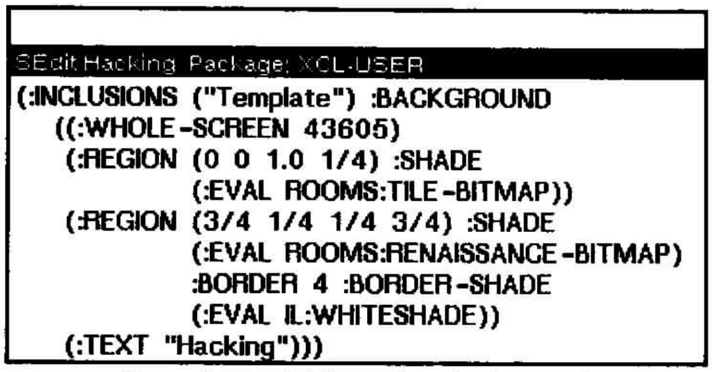
Figure 3-20. A room with two regions specified

Putting a Border Around a Region

You can put a border around a region using `:BORDER` and specify a border shade using `:BORDER-SHADE`.

`:BORDER`	in pixels; defaults to zero (0).
`:BORDER-SHADE`	a shade; defaults to `IL:BLACKSHADE`.

You specify a border and shade for a region by putting these keywords inside a `:REGION` argument to `:BACKGROUND` (Figure 3-21).


SEdit Hacking Package: XCL-USER

```
(:INCLUSIONS ("Template") :BACKGROUND
   ((:WHOLE-SCREEN 43605)
    (:REGION (0 0 1.0 1/4) :SHADE
             (:EVAL ROOMS:TILE-BITMAP))
    (:REGION (3/4 1/4 1/4 3/4) :SHADE
             (:EVAL ROOMS:RENAISSANCE-BITMAP)
             :BORDER 4 :BORDER-SHADE
             (:EVAL IL:WHITESHADE))
    (:TEXT "Hacking")))
```

Figure 3-21. Adding a border to a region

:TEXT—Writing Text in the Background

:TEXT lets you write text anywhere on the screen. The text written becomes part of the background; that is, it cannot be edited and can only be changed by modifying the room definition. When a room is first created :TEXT has as its default value the name of the room with the text placed in the lower left corner of the room.

:TEXT has one mandatory and four optional arguments. The mandatory argument is the text itself. The optional arguments are: the position of the text on the screen (:POSITION), the relative relationship between the position and the text (:ALIGNMENT), the font of the text (:FONT), and the text's shadows (:SHADOWS).

Note: You can use :TEXT as many times as you want in a background description but if you don't specify different positions all the text is written one on top of the other in the lower left corner.

:POSITION provides the position of the text, given in the dotted pair notation of (LEFT . BOTTOM). When you specify :POSITION you can use integers and floating point numbers. You may also specify a position using fractions to specify a relative position. For example, a position of (0 . 500) places the text on the left hand side, a little over halfway up the height of the screen. A specification of (1/2 . 1/2) places the text at a position half the screen width from the left and half the screen height from the bottom.

Your screen size can be determined from the values of IL:SCREENWIDTH and IL:SCREENHEIGHT.

:ALIGNMENT gives the relationship of :POSITION to the text. If the text has shadows, the alignment is calculated using the shadows. Its value is one of :LEFT-BOTTOM (default), :LEFT-TOP, :CENTER, :RIGHT-BOTTOM or :RIGHT-TOP. For example, to write text in the lower right-hand corner of the screen, you would use :RIGHT-BOTTOM. Remember, it's the position in relation to the text, *not* the text in relation to the position (Figure 3-22).

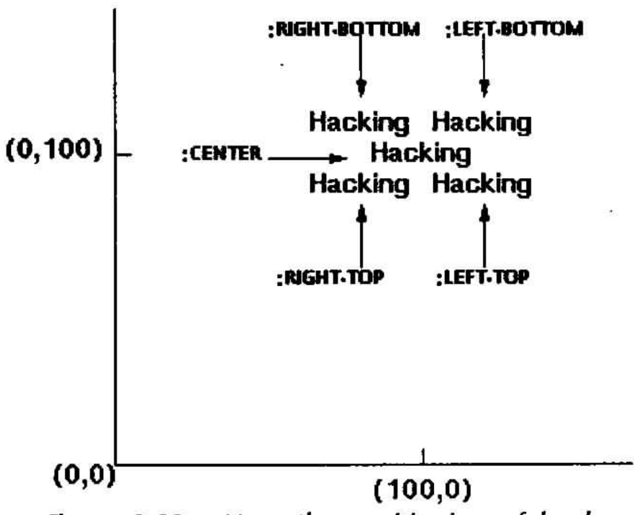


Figure 3-22. How the positioning of background text responds to changes in :ALIGNMENT

`:FONT` specifies the font of the text. Given in the form of (FAMILY SIZE FACE). For example text fonts might be, (MODERN 18 BRR) or (CLASSIC 24 MRR). Defaults to the value of: **`*default-background-text-font*`** which is initially 36 point, TimesRomand medium.

Note: For information on how to change **`*default-background-text-font*`** see Appendix A, Programmer's Guide to ROOMS, "Room Backgrounds."

Sample uses of the `:TEXT` keyword:

```
(:TEXT "Sample Text" :POSITION (500 . 500)
:ALIGNMENT :CENTER :FONT (CLASSIC 24 MRR))
```

Writes the string "Sample Text" centered near the middle of the screen.

```
(:TEXT "Help String" :POSITION (500 . 800))
```

Writes the string "Help String" centered near the top of the screen.

`:SHADOWS` specifies shadows for the text written on the background. Usually one of:

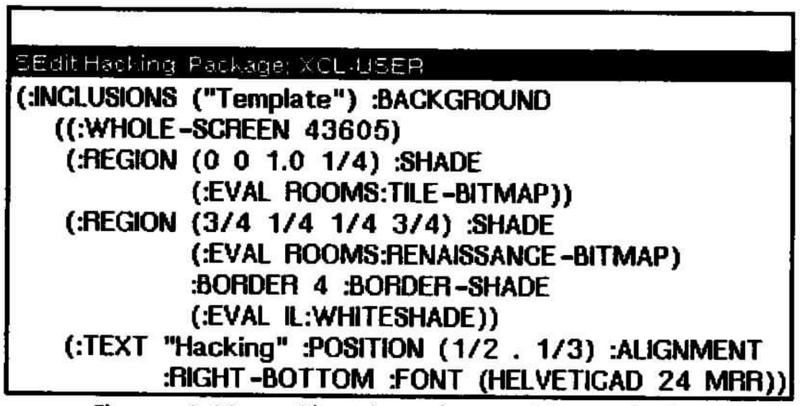
T Specifies that shadows are present.

NIL Specifies that shadows are not present.

The default value is T.

See Appendix A, Programmer's Guide "Text Shadows" for how to specify other shadows.

Thus, you can move the text around in Hacking using :TEXT and its arguments (Figure 3-23). You can change the text entirely if you wish but the name of the room remains the same.



```
(:INCLUSIONS ("Template") :BACKGROUND
  ((:WHOLE-SCREEN 43605)
   (:REGION (0 0 1.0 1/4) :SHADE
            (:EVAL ROOMS:TILE-BITMAP))
   (:REGION (3/4 1/4 1/4 3/4) :SHADE
            (:EVAL ROOMS:RENAISSANCE-BITMAP)
            :BORDER 4 :BORDER-SHADE
            (:EVAL IL:WHITESHADE))
   (:TEXT "Hacking" :POSITION (1/2 . 1/3) :ALIGNMENT
          :RIGHT-BOTTOM :FONT (HELVETICAD 24 MRR))
```

Figure 3-23. Changing the position of the background text

[This page intentionally left blank]

4. PLACEMENTS

ROOMS supports windows appearing in more than one room by using placements. Placements are structures that maintain information on a window (in the form of a pointer) and on a window's location in a particular room. There is a separate placement for every window in every room allowing ROOMS to display the same window using different regions in different rooms.

For example, you may want to have an Exec in every room. However, you probably don't want a *different* Exec for every room as this would require more system overhead, in the form of many Exec processes, than you might want to incur. Therefore, you need the ability for several rooms to share the same window; placements provide this.

For a more technical explanation of placements see Appendix A, Programmer's Guide to ROOMS, "Placements."

Manipulating Placements

Since window functionality is only enhanced by ROOMS, all of the standard window functions still apply (for example, closing a window and moving a window around on the screen). In ROOMS these functions apply to placements. The window manipulation functions described below are additional features that are executed within the rooms themselves. The Overview allows for additional window functionality; those commands are described in Chapter 6, Overview.

Move a Placement From One Room to Another

ROOMS allows you to move a placement from one room to another using the standard right mouse button window menu (Figure 4-1).

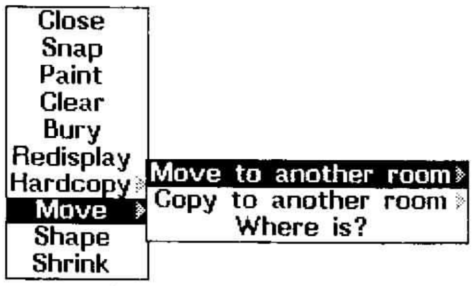


Figure 4-1. The window menu command to move a placement to another room

After you select "Move to another room," ROOMS opens a menu with all the existing rooms on it (Figure 4-2). If you select <new room> you are prompted for the name of the new room, it is created and then the placement is moved to the new room. Using any mouse button select the room to which you want to move the placement. ROOMS moves the placement into the room that you select, puts the placement in the same screen location as in the current room, and deletes the placement from the current room.

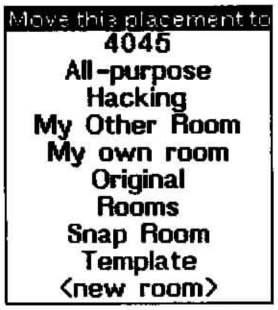


Figure 4-2. The "Move this placement to" menu

Another way to move a placement is by using Pockets; select "Move to pockets" (Figure 4-3). A placement in Pockets automatically engenders a placement in every room. Thus, a placement you move from the current room to Pockets doesn't disappear. Rather, ROOMS moves the placement to Pockets at the current screen position, deletes the placement from the current room, then propagates the placement throughout all rooms. The window appears not to move at all.

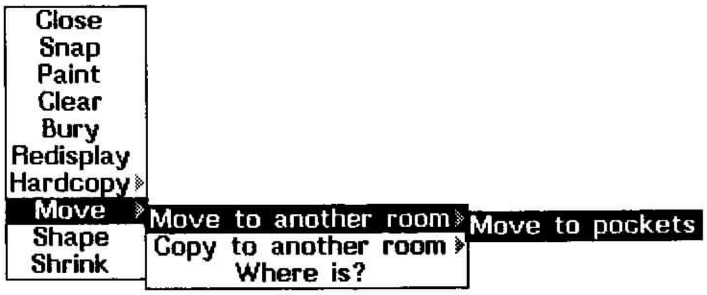


Figure 4-3. The "Move to pockets" submenu

Copy a Placement From One Room to Another

ROOMS allows you to create a copy of a placement and put it into another room using the right mouse button window menu (Figure 4-4).

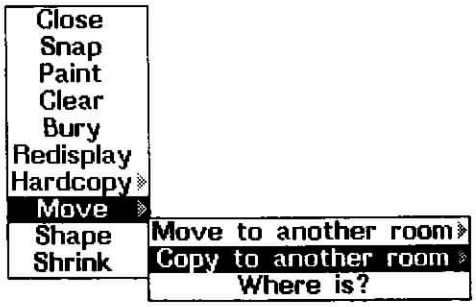


Figure 4-4. The "Copy to another room" command

When you select "Copy to another room" the "Copy this placement to" menu opens listing all existing rooms (Figure 4-5). Select the room in which you want to put a copy of the placement. ROOMS creates a copy of the placement and puts it into the room you selected (at the same screen location as it had in the current room).

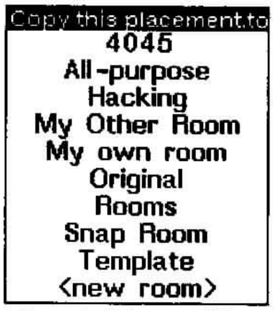


Figure 4-5. A "Copy this placement to" menu

You can also make a copy of a placement in the current room if the placement does not originate there. For example, you may have a set of placements in room "A" (called the destination room) that originate from room "B," (called the source room) i.e., you have *included* room "B" in room "A." In fact, let's say that you have included room "B" in two other rooms, "C" and "D".

It is a property of included placements that they have the same position in all the rooms they are included in. Thus, an Exec positioned in the upper left corner of room "B" (the source room) will also appear in the upper left corner of rooms "A," "C," and "D." Moreover, if you move that Exec to another location in *any* of the four rooms, it changes location in *all* the rooms.

If you want the included Exec to appear in the lower right corner of room "A" but remain untouched in rooms "B," "C," and "D" you would:

1. Go to room "A."
2. Make a copy of the included Exec by selecting "Copy to this room" (Figure 4-6).
3. Move the included Exec to the lower right corner of room "A."

The included Exec stays in the upper left corner of rooms "B," "C," and "D."

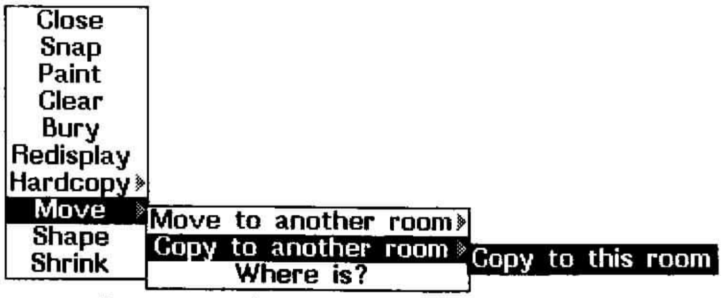


Figure 4-6. The "Copy to this room" submenu

Closing a Placement

Since a placement can appear in more than one room, a conflict arises when you attempt to close a placement in one room that has copies in others. That is, ROOMS needs to know: do you want to close only this copy of the placement or do you want to close every existing copy of that placement in the ROOMS environment?

Closing a placement removes the *placement* from a room. A *window* is closed only when *all* the placements are closed. Closing all the placements is the same as closing a window using the right mouse button window menu.

Let's continue with the example from "Copy a Placement" above.

When you close the copy of the Exec in room "A" ROOMS opens a menu asking whether or not you want to close just the placement in room "A" or the placements in all the rooms where it occurs (Figure 4-7).

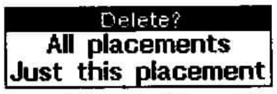


Figure 4-7. The "Delete?" placement menu

Selecting "All placements" closes every instance of the placement. Selecting "Just this placement" closes only the placement in the current room.

Note: Pressing a mouse button with the cursor outside the "Delete?" menu aborts the deletion operation.

So, if you choose to close just the placement of the Exec in room "A," the Exec in the lower right corner disappears. However, the Exec opens in the upper left corner. This Exec is in room "A" by virtue of being in the included room "B."

When you close an included placement ROOMS asks you to confirm the action before the placement is closed. The message, "This placement is in the included room "<room name>". Are you sure you want to delete it?" prints in the prompt window. Press the left mouse button to confirm closing the placement. Press any other mouse button to abort the operation.

"Where is?"—Finding Which Room a Placement Is In

After you've copied a few placements to the current room you may need to find out to which room a placement actually belongs, especially if you've included several rooms in the current room. ROOMS provides a "Where is" facility for finding out this information.

To use the "Where is" facility, open the right mouse button window menu and select "Where is?" from the Move submenu (Figure 4-8). ROOMS prints a message in the prompt window telling you the location of the actual placement for the window.

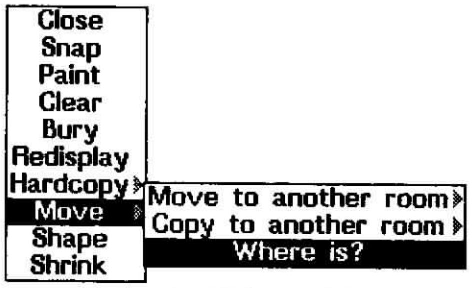


Figure 4-8. The "Where is?" command

A window that appears in more than one room because you specifically copied it to each one has an actual placement in each room.

Modules and Applications that Reuse Windows

Modules and applications that reuse windows (the Library module Chat is a notable example) can be induced to start in a room other than the current room. For example, if you open a Chat window in room "A"; quit Chat, but leave the window open. Go to room "B". Restart Chat: the Chat window in room "A" is activated but is not brought to the current room. This occurence is a general effect of the interaction between ROOMS and modules and applications that reuse windows.

If you want to prevent this from happening close windows that are reused when you're done with them. Then, when you reactivate the application or module, the window opens in the current room.

The "Retrieve Windows" Command

If an error occurs, you could have a placement in a room that is not visible when you enter that room; the placement is considered "lost." If you believe that you have some windows that should be visible but are not, select "Retrieve Windows" from the ROOMS submenu (Figure 4-9).

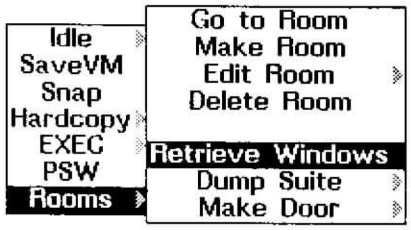


Figure 4-9. The "Retrieve Windows" command

Note: It is very hard to "lose" windows just by using ROOMS at the menu level. In general you probably won't need to use "Retrieve Windows" unless you are manipulating ROOMS programatically.

All windows that ROOMS determines are not assigned to any room are brought into the current room. If ROOMS detects no unassigned windows then the message "All windows are in some room." prints in the prompt window.

5. NAVIGATION

ROOMS provides several aids to help you navigate between rooms and move windows from one room to another. Using the ROOMS submenu is one way to move around; using buttons are another.

Buttons are a unique user-interface device that provide for the execution of commands at the click of the mouse. One way to move between rooms is through doors. Doors are just buttons that take you to another room. Also, using the baggage utility, you can take windows with you when you move between rooms.

For complete information on the use of buttons as other than doors see Appendix A, Programmer's Guide to ROOMS, "Buttons."

"Go to Room" Menu Option

One way to go to another room is by selecting "Go to Room" from the ROOMS submenu (Figure 5-1) .

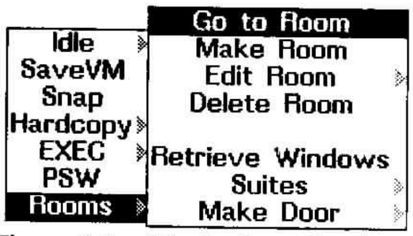


Figure 5-1. "Go to Room" selected

When you release the right mouse button the "Go to room" menu opens listing all the known rooms and a "new room" item that allows you to create a room and then immediately go to it (Figure 5-2). Select the room you want to go to using any mouse button.

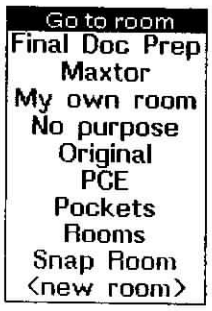


Figure 5-2. A "Go to room" menu

Doors

A door is a specialized placement that takes you from one room to another room with the click of a mouse button. A door takes you to the room whose name is printed on it. See Figure 5-5 for the way the standard doors look.

Creating Doors

You create doors using the ROOMS submenu (Figure 5-3). Door creation is described below; manipulating and changing doors is described in Appendix A. The Programmer's Guide to ROOMS, "Buttons."

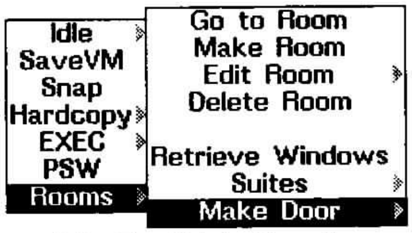


Figure 5-3. The "Make Door" command

Selecting "Make Door" opens a menu that is a list of all the existing rooms and a "new room" item that allows you to create a room and a door to it at the same time (Figure 5-4).

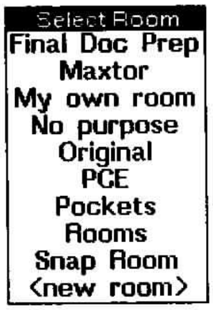


Figure 5-4. The "Select Room" menu

Using any mouse button, select the room for which you want a door. After you select the room the "Select Button Type" menu opens with all the known button types (Figure 5-5). This menu also illustrates all the standard button types.

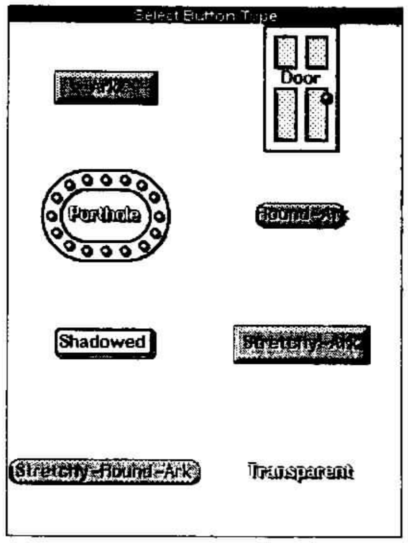


Figure 5-5. The "Select Button Type" menu for door creation

Selecting one of the button types causes the button you chose to appear (in outline form) at the position of the cursor. Move the outlined door to the position on the screen where you want it and press the left mouse button. The button, with the name of the room you selected printed on it, appears.

Doors in general and Back Doors in particular can have the name of a room that no longer exists. ROOMS does not automatically delete doors when you delete a room. If you try to use a door to a nonexistent room ROOMS prints the message: "No room named "<room name>" exists."

Note: You can define your own button types. See Appendix B, Programmer's Guide to ROOMS, "Buttons."

Manipulating Doors

Doors are contained in windows and selecting a door using the right mouse button opens the standard window menu. Using this menu you can move, copy, and close doors in the same way you can other placements (see Chapter 4, Placements). You can also use the MOVE and COPY keys to manipulate doors.

Using the MOVE Key

Holding down the MOVE key while you select a door using the middle mouse button has the same effect as selecting "Move" on the window menu.

After you select the door you want to move you can release the MOVE key (but keep the mouse button down). When the door is positioned where you want it, release the middle mouse button.

Using the COPY Key

Holding down the COPY key while you select a door using the middle mouse button creates a copy of the door (If the TTY is in a TEdit or Sketch window a button image object is inserted into the TEdit or Sketch window at the point of the caret. See the section "Creating and Using Button Image Objects" below for more information on button image objects.)

When you use the COPY key to copy a door whose placement originates in another room no message prints in the prompt window as it does when you use the "Copy to another room" command.

Using the DEL(ETE) Key

Holding down the DEL(ETE) key while you select a door using the middle mouse button deletes the door from the room.

Back Doors

After entering a room, it is quite natural to want to go back to the room that you just came from. If you don't remember the name of the previous room this could result in a time-consuming search. ROOMS provides back doors that return you to the previous room. You can make back doors by selecting "Make Back Door" (Figure 5-6).

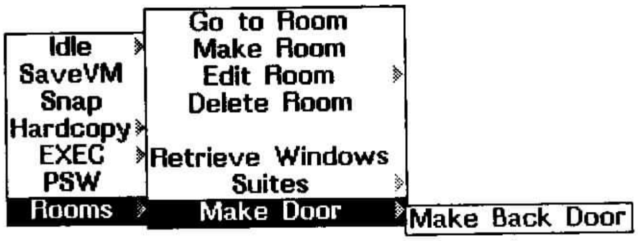


Figure 5-6. The "Make Back Door" submenu

Selecting "Make Back Door" creates a special door (Figure 5-7). The door is inverted to differentiate it from other types of doors. When you select a back door with either the left or middle mouse button, you go to the room that you were last in. In addition, ROOMS updates the door on every future entrance to the room so that it always takes you to the last room you were in.

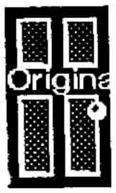
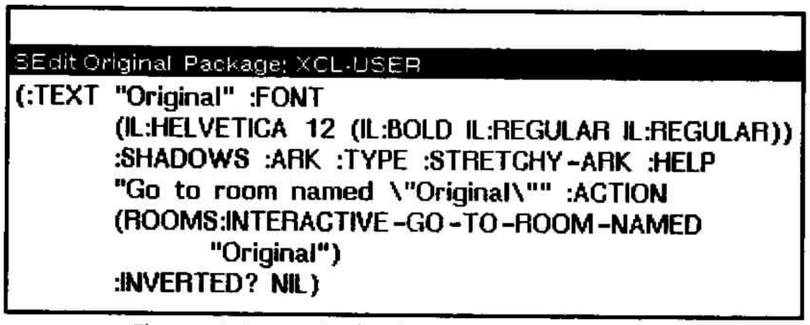


Figure 5-7. A back door

Changing a Door's Appearance

While it is probably easier to just create a new door than change the description of an existing one, you can edit a door. To change the description of a door, hold down the EDIT (or PROPS) key while selecting it using middle mouse button. This opens an SEdit window with that door's description (Figure 5-8).



```
(:TEXT "Original" :FONT
       (IL:HELVETICA 12 (IL:BOLD IL:REGULAR IL:REGULAR))
       :SHADOWS :ARK :TYPE :STRETCHY-ARK :HELP
       "Go to room named \"Original\"" :ACTION
       (ROOMS:INTERACTIVE-GO-TO-ROOM-NAMED
             "Original")
       :INVERTED? NIL)
```

Figure 5-8. An SEdit description of a door

You can find the meaning of the various keywords in a door's description in Appendix A, Programmer's Guide to ROOMS, "Buttons" (in the description of `make-button`).

Baggage

Baggage is a utility that lets you carry windows with you as you travel from one room to another. When you are carrying windows as baggage you can either take a copy with you and delete the original (MOVE) or take the copy and leave the original alone (COPY).

There are two ways to use the baggage utility:

1. Hold down the MOVE or COPY key as you select a door using the left mouse button. The cursor changes to a cross-hairs (⊕); using the left mouse button select the window(s) that you want to take with you.

When you are finished selecting windows, either click with the left mouse button on the background or release the MOVE or COPY key before you select the last window. ROOMS moves you through the door and the windows that you selected go with you.

2. Use only the MOVE key, interacting with the ROOMS "Go to Room" menu selection.

   a. Open the background menu.

   b. Open ROOMS submenus until you can select "Go to Room."

   c. When the "Go to Room" menu listing all the rooms in the environment opens press and hold the MOVE key.

d. Select the room you want to go to.

From this point on the behavior is the same as above, to wit: the cursor changes to cross-hairs indicating that you are to select, using the left mouse button, the window(s) you want to take with you.

When you are finished selecting windows, either press the left or middle mouse button on the background or release the MOVE key before you select the last window. ROOMS moves you to the room you selected from the menu. The windows that you selected go with you.

Even if you don't select any windows when you press the left (or middle) mouse button with the cursor in the background, you go to the room you chose.

Creating Non-Door Buttons

ROOMS allows you to create buttons that are not doors and which can take the place of selecting ROOMS commands from the ROOMS submenu.

For example you can create a "Go to Room" button that operates identically to the "Go to Room" menu command.

1. Open the background menu.
2. Select ROOMS from the background menu but do not open the submenu.
3. Press and hold the COPY key.
4. Open the ROOMS submenu. You should still have the COPY key held down.
5. Select "Go to Room" from the submenu and release the right mouse button.
   - An outline of a button appears at the position of the cursor.
6. Release the COPY key.
7. Position the button's outline where you want it on the screen and press the middle or left mouse button.
   - A "Go to Room" button appears. Selecting this button (with the left or middle mouse button) opens the "Go to room" menu listing all the existing rooms.

This procedure works for all the ROOMS submenu commands except "Make Door" and "Make Back Door." Selecting "Rooms" from the background menu creates a button that takes you to the Overview.

For complete information on making and using non-door buttons see Appendix A, Programmer's Guide to ROOMS, "Buttons."

Creating and Using Button Image Objects

You can create button image objects to put into TEdit or Sketch files (For information on Interlisp image objects please consult the *Interlisp-D Reference Manual*, Chapter 27, "27.16 Image Objects.")

To create button image objects do the following:

1. Give the TTY to either Sketch or TEdit (i.e., the caret should be in a TEdit or Sketch window).
2. Press and hold the COPY key.
3. Select a button using the middle mouse button.
4. Release the COPY key.

The fully active button, as a button image object, is inserted into the TEdit or Sketch window. Once the button image object is in TEdit or Sketch, selecting it using the left mouse button activates the button's action. Selecting it with the middle button opens a menu (Figure 5-9).

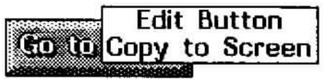


Figure 5-9. The button image object menu

When you select "Edit Button" from the menu an SEdit window opens with the button's description; select "Copy to Screen" and a copy of the button is created. Press the left or middle mouse button when the copy is where you want it on the screen.

Remember, if the TTY is not in a TEdit or Sketch window then pressing the COPY key while selecting a button using the middle mouse button makes a copy of the button to be placed on the background.

[This page intentionally left blank]

6. THE OVERVIEW

The Overview lets you view all existing rooms at once. From the Overview you can perform operations on both placements and rooms (e.g., copy and move placements between rooms, edit rooms, etc.), and look at the contents of any room.

What the Overview Looks Like

The Overview lays out every room in your ROOMS environment, in alphabetical order, through the use of pictograms. In the Overview a pictogram is used as an active, shrunken representation of a room and all it contains.

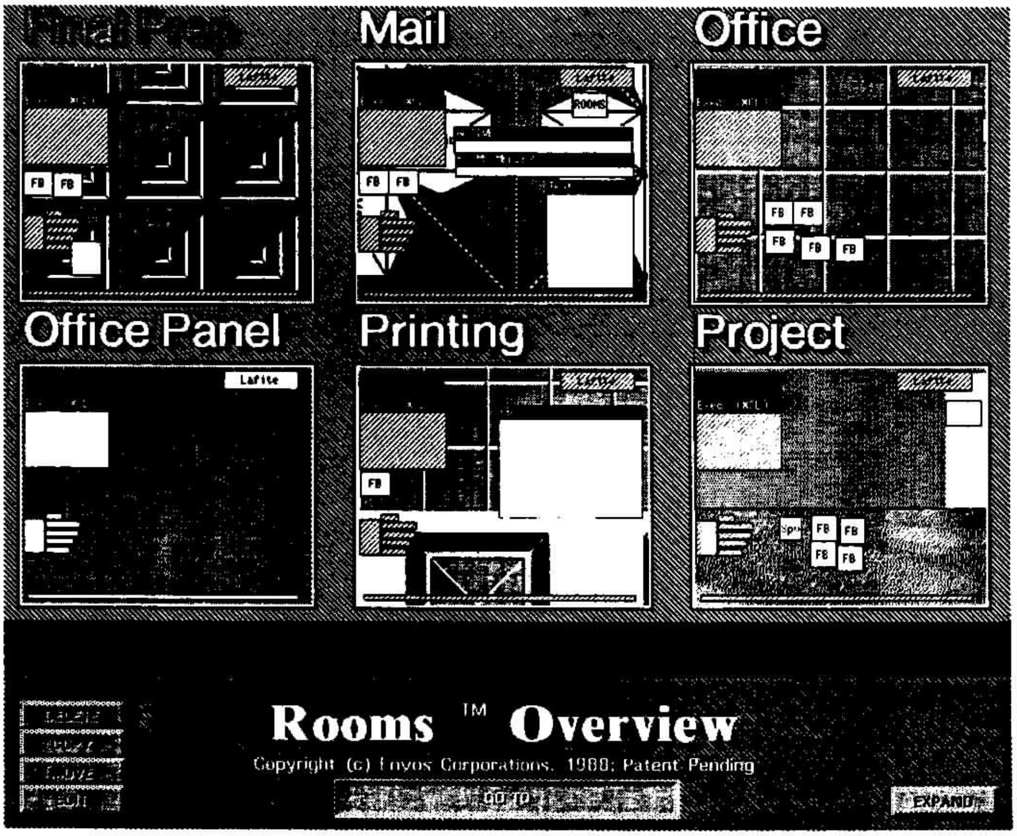


Figure 6-1. A ROOMS Overview

You enter the Overview by selecting "Rooms" from the background menu. When you enter the Overview the name of the room you were last in is in gray letters; all other room names are in white letters.

Note: A pictogram reflects the state of a room after its last update. A room is updated whenever it is exited.

How the Overview Works

Getting things done in the Overview relies on the coordinated use of the mouse and the keyboard. Since you can operate on both placements and rooms in the Overview there are aids and conventions to follow to help you accomplish your task.

In order to differentiate between operations intended to affect placements and those directed at rooms, the Overview uses the following conventions:

Left mouse button:	selects placements
Middle mouse button:	selects rooms

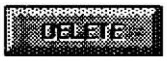
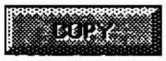
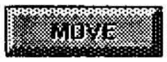
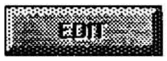
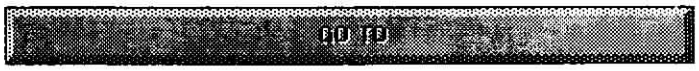
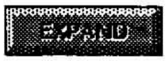
Six keyboard keys are used together with the mouse to operate on placements and rooms. In general, the keys do what they are labeled, that is the DEL(ETE) key deletes rooms and placements, the COPY key copies them, etc. See Table 6-1 for the correlation between the bitmap representations, the keyboard keys, and the actions taken.

Note to Sun Workstation Users: Figure 6-1 represents how the Overview is laid out on Xerox 1186 and 1108/9workstations. If you are running the Envos Medley 1.0-S on a Sun workstation the keyboard is laid out differently and the Overview you see reflects those *spatial* differences. All functionality remains the same. For information on how the Sun keyboard is laid out for Medley 1.0-S please consult the *Medley 1.0-S User's Guide*.

The bitmap representations of the keys used in the Overview are at the bottom of the Overview; they are mouse sensitive. Pressing one of the keys or selecting the bitmap using either the left or right mouse button activates each operation. Each key selected remains active until you either select another key or leave the Overview.

When you use a mouse-button-key combination that the Overview does not recognize (like attempting to edit a placement), ROOMS prints a message in the prompt window.

Table 6-1. Mapping of Overview key bitmaps to the keyboard

Key	Bitmap Representation	Placement Operation	Room Operation
DEL(ETE)		Deletes	Deletes
COPY		Copies/Prompts for positioning	Copies/Prompts for name
MOVE		Moves/Prompts for positioning	Copies/Deletes old/Prompts for name
EDIT		None	Opens SEdit window
SPACE BAR			Enters room (preselected when Overview entered)
EXPAND		Brings representation of placement into Overview	None

Using the Overview

Manipulating Rooms

This section tells you how to manipulate rooms from the Overview. All selections are done using the middle mouse button.

Enter a Room

To enter a room, press the space bar and select the room you want to enter. ROOMS moves you to that room. When you first enter the Overview the space bar is pre-selected.

Copy a Room

To copy a room:

1. Press COPY and select the room you want to copy.
   - ROOMS prompts you (in the prompt window) for the name of the new room.
2. Type the name of the new room and press <RETURN>.
   - The new room, an exact replica of the selected room, is added to your environment.

If you give the new room the same name (the names are case sensitive) as an existing room, ROOMS tells you that a room of that name already exists and the copy command is cancelled.

Delete a Room

To delete a room:

1. Press DEL(ETE).
2. Select the room you want to delete.
   - You are asked to confirm the deletion.
3. Press the left mouse button to confirm the deletion.

Pressing any other mouse button during the confirmation aborts the deletion process.

Rename a Room

To rename a room:

1. Press MOVE.
2. Select the room you want to rename.
3. Enter the new name in the prompt window and press <RETURN>.

The room you selected is renamed.

Edit a Room

To edit a room:

1. Press EDIT (or PROPS).
2. Select the room you want to edit.

ROOMS opens an SEdit window containing the room description. If you make any changes, when you end the editing session both the room and the pictogram are updated to reflect them.

Manipulating Placements

This section tells you how to manipulate placements from the Overview. All selections are done using the left mouse button.

Copy a Placement

To copy a placement:

1. Press the COPY key.
2. Select the placement you want to copy.
    - The cursor changes to a crosshairs (⊕).
3. Move the cursor until the new placement is positioned where you want it (either in the same or a different room).
4. Press the left mouse button.
    - The placement is copied to the new position.

If you press the left mouse button when the cursor is not in any room (after you have selected a placement to copy), the error message "Invalid destination" prints in the prompt window and the command is cancelled.

Copying a placement to or from a room that is included in other rooms automatically propagates the change to all rooms that include that room (see Chapter 3, Rooms, ":INCLUSIONS").

Move a Placement

To move a placement from one room to another or within a room:

1. Press the MOVE key.
2. Select the placement that you want to move.
    - The cursor changes to a crosshairs (⊕).
3. Move the cursor until the placement is positioned where you want it (either in the same or a different room).
4. Press the left mouse button.
    - The placement is moved to the new position.

If you press the left mouse button when the cursor is not in any room (after you have selected a placement to move), the error message "Invalid destination" prints in the prompt window and the command is canceled.

Moving a placement into (or from) a room that is included in other rooms automatically propagates the change to all rooms that include that room (see Chapter 3, Rooms, ":INCLUSIONS").

Expand a Placement

You can bring the small representations of placements that you see in the pictograms into the Overview, and expand them to the size they actually have in the rooms.

1. Press EXPAND.
2. Select the placement you want to expand.

The placement appears in the Overview, of the same size and at the same screen position that it has in the room. Pressing the left mouse button again closes the expanded placement.

Delete a Placement

To delete a placement:

1. Press the DEL(ETE) key.
2. Select the placement that you want to delete and press the left mouse button.
    - That placement is deleted.

If you press the left mouse button when the cursor is not in any room (after you have pressed the DEL(ETE) key), nothing happens.

Deleting a placement in a room that is included in other rooms automatically propagates the change to all rooms that include that room (see Chapter 3, Rooms, ":INCLUSIONS").

Deleting a placement removes the *placement* from a room. A *window* is closed only when *all* the placements are deleted. Deleting all the placements is the same as closing a window using the right-button window menu.

Note: If you attempt to delete a type of window that requires you to confirm the deletion (e.g., a TEdit window with unsaved changes or a file browser with files marked for deletion) you are asked to take that confirming action before ROOMS deletes the window.

Editing Placements Outside the Overview

You can edit the placements, using the placement editor, for any room without going to that room or the Overview. When you select "Edit Placements" (Figure 6-2) a menu listing all the rooms (including ones in suites) opens (Figure 6-3). Select the room whose placements you want to edit. A pictogram, similar to what you see in the Overview, opens in the room you are in; you can see the placement arrangement for the selected room (Figure 6-4).

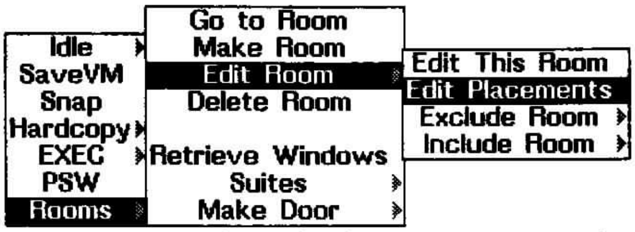


Figure 6-2. The "Edit Placements" command

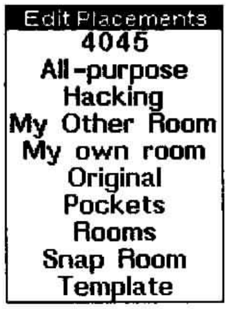


Figure 6-3. An example of an "Edit Placements" menu

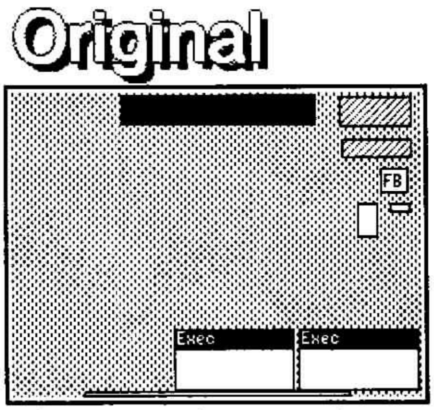


Figure 6-4. A pictogram of a room

Using this pictogram, you can do the same operations on placements as in the Overview but only *within* the named room.

With the cursor in the "title bar" of the pictogram (when not in the Overview) pressing the middle mouse button opens the "ReFetch" menu (Figure 6-5). Select "ReFetch" and the placements for that room are updated (See Chapter 4, Placements).

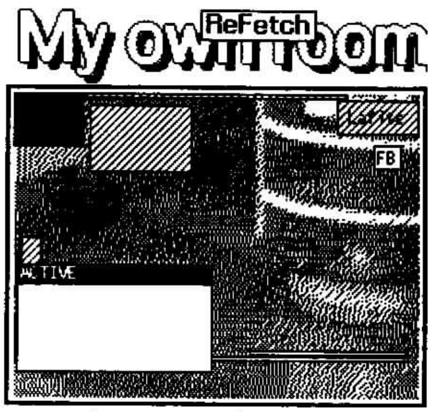


Figure 6-5. Placement editor with the "ReFetch" menu open

When you close a room's pictogram, you are prompted with a menu asking whether you want to close all placements for this window. Select either option, only the pictogram opened by the placement editor is closed.

[This page intentionally left blank]

7. SUITES

A suite is a grouping of rooms that you can save and load like any Envos Lisp file. ROOMS does not allow any rooms to have the same name so you are encouraged *not* to include the same room in more than one suite.

Note: You cannot include either Original or Pockets in a suite.

Manipulating Suites

When a suite is present, those rooms that are part of the suite are treated as a single unit for certain operations. For example, you cannot delete a room that is part of a suite without specifically removing it from that suite first.

The commands described below are found on the Suite submenu (Figure 7-1).

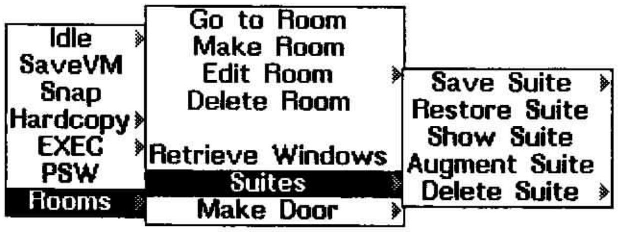


Figure 7-1. The ROOMS "Suite" command and the suite submenu. Selecting "Suites" opens the suite submenu.

"Save Suite"—Creating and Saving a Suite

The "Save Suite" command is used to save a set of rooms onto a file. It can also be used to create a new suite. The suite file is written to the connected directory. Selecting "Save Suite" opens the "Save" menu which is a list of suites if any exist. If no suite exists then the menu contains the single item "<new suite>" (Figure 7-2).

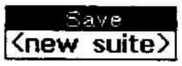


Figure 7-2. The "Save" menu as it appears if no suites are present

Creating a New Suite

To create a suite and then immediately write the new suite to a file follow these steps:

1. Select "<new suite>" from the "Save" menu.
    - If all the existing rooms are part of some suite ROOMS prints the message, "All rooms are already in some suite." in the prompt window. The command is cancelled.

2. Enter the name of the suite in the prompt window and press <RETURN>.

   - A menu opens listing all the rooms (except Pockets and Original) that are not in any suite (Figure 7-3).

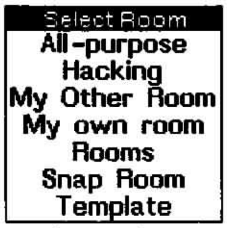


Figure 7-3. The "Select Room" menu

3. Select the first room you want to add to the suite using any mouse button.

   - The "Select Room" menu closes and reopens with the room you selected to be in the suite deleted from the menu.

4. Continue selecting rooms to add to the suite. Each one is deleted in turn from the "Select Room" menu.

5. When you have selected all the rooms you want in the suite press any mouse button outside the menu.

Note: Once you add a room to a suite, you can't delete that room from the environment until you have removed it from the suite (see "Delete Room From Suite" below).

   - ROOMS prints a series of messages in the prompt window telling you of its progress:

     "New SUITES definition for <suite name> (but not installed)."

     "Making file <suite name> . . ."

     "Made file <connected directory> <suite name>."

Once you have the suite file saved on your disk or a file server, loading the suite recreates the set of rooms and their placements. Suites obviate the need to recreate a standard set of rooms each time you load a fresh sysout.

For each room in a suite that you want to save ROOMS goes through and tries to abstract every window. However, each window must have a pre-existing window type definition so that ROOMS knows how to save and recreate that window. If ROOMS doesn't recognize a type of window or can't save a type it knows about (like SEdit windows) then the window is brought to the current room and flashed several times. Also, a message is printed in the prompt window.

The ROOMS software includes window type definitions for the standard Envos Lisp windows: prompt, Exec, TEdit, Logo, and FileBrowser. See Appendix A, Programmer's Guide to ROOMS "Window Types" for more information on defining your own window types.

"Restore Suite"—Loading a Suite File

Loading a suite file into a ROOMS environment adds the rooms on the file to your existing set of rooms. To load a suite, select "Restore Suite" from the background menu (Figure 7-1). Type the name of the suite you want to load in the prompt window and press <RETURN>.

ROOMS looks for the suite file using the directory list **`*suite-directories*`**, a list of pathnames. The initial value is **`(t)`** which is the connected directory (see the *Interlisp-D Reference Manual*, Chapter 24, Streams and Files "Section 24.16 Searching File Directories").

Note: Loading a suite file that was made with the "Save Suite" command automatically loads the ROOMS software if it isn't loaded.

If you attempt to load a suite that is already present in the current ROOMS environment the message, "A suite named <name> is already loaded," prints in the prompt window. The command is ignored.

Room Name Conflicts

If you try to load a suite with a room that has the same name as one that already exists, ROOMS causes a debugger window to open. This protects you from automatically losing any existing room(s) (and all the windows) by having the existing room replaced with a new one.

There are two ways to continue when this occurs:

1. Abort the suite load

Press the middle mouse button on the debugger window and select the ↑ (up-arrow) option. This aborts the suite load, letting you resolve the name conflict at a later time. ROOMS checks for name conflicts before doing anything else. So, if you abort the suite load, your environment is in the same state as before the load.

Each time during the suite load that ROOMS encounters a name conflict, a debugger window opens. If you abort out (↑) of *any* room conflict debugger the entire suite load is cancelled.

2. Delete the existing room

Press the middle mouse button on the debugger window and select the proceed option (Figure 7-4).

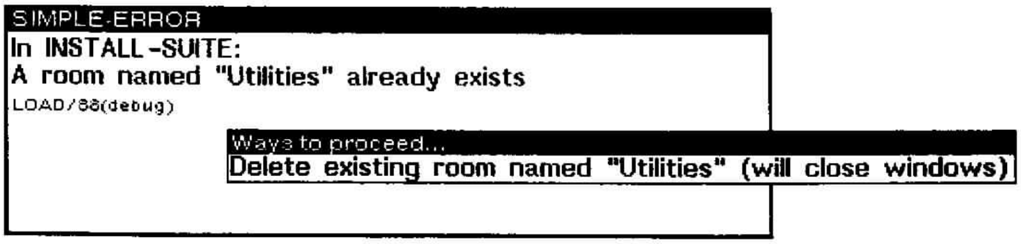

```
SIMPLE-ERROR
In INSTALL-SUITE:
A room named "Utilities" already exists
LOAD/88(debug)
```

Figure 7-4. Suite debugger window with the proceed menu open

The existing room is deleted, the placements within that room are deleted and the new room is added to your environment as part of the newly loaded suite.

"Delete Suite"

At some point after loading a suite you may want to remove the suite while leaving the rooms in it untouched. Or you may want to delete the suite and all the rooms it contains. Select "Delete Suite" for these purposes.

After selecting "Delete Suite," ROOMS responds with a menu of all the known suites (Figure 7-5). Select the suite you want to delete using any mouse button. Press the left mouse button to confirm the deletion. Deleting a suite simply ungroups all of the rooms so that they are treated as separate entities rather than as part of a suite.

Pressing the middle or right mouse button aborts the deletion operation.

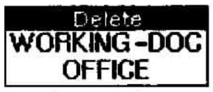


Figure 7-5. The "Delete" suite menu

After deleting the suite ROOMS asks if you also want to delete all the *rooms* in the deleted suite. Press the left mouse button to confirm the room(s) deletion. Pressing any other mouse button aborts the operation.

Deleting the room(s) in the deleted suite closes all the windows in the suite that are not in any room *outside* of the suite.

"Delete Room From Suite"

"Delete Room From Suite" (Figure 7-6) allows you to disassociate a room from a suite. You may want to remove a room from a suite in order to add it to another suite or to delete the room from the environment. You can't delete a room that is part of a suite until you remove the room from its suite first.

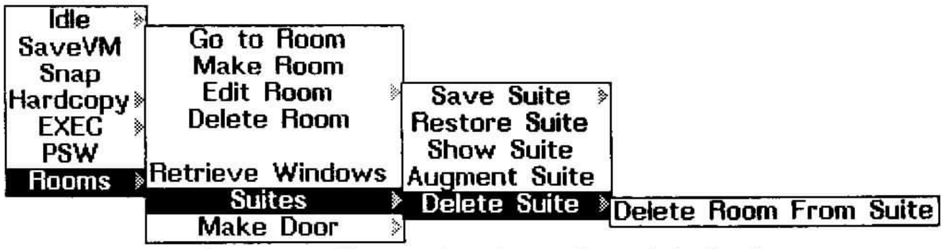


Figure 7-6. The "Delete Room From Suite" submenu

First the "Delete room from" menu (Figure 7-7) opens listing all the existing suites. Select the suite using any mouse button. Pressing a mouse button outside the menu aborts the operation.

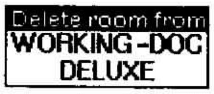


Figure 7-7. An example of the "Delete room from" menu

After you select the suite the "Select Room" menu (Figure 7-8) opens. This menu is a list of all the rooms in the suite you selected from the previous menu. The room you select from this menu is disassociated from the suite.

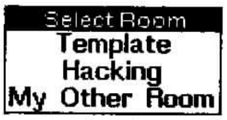


Figure 7-8. An example of a "Select Room" menu

"Update Suite"

Select "Update Suite" (Figure 7-9) when you want to make a suite reflect its current state but you don't want to write it to a file. When you select "Save Suite" ROOMS does a suite update before the suite file is written out. Also, once you have one or more suites present you can use "Update Suite" to create a new suite.

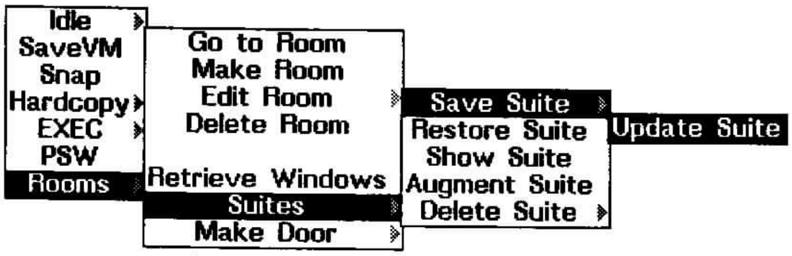


Figure 7-9. The "Update Suite" Menu

After you select "Update Suite" the "Update" menu (Figure 7-10) opens listing all the known suites. Select the suite you want to update.

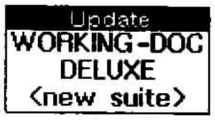


Figure 7-10. An example of an "Update" menu

After you select the suite to update ROOMS goes through the entire suite and updates it to reflect any changes (new placements, new rooms added, etc.) made between the last save and the current version of the room.

Updating a Suite and the File Manager

When ROOMS finishes updating a suite the message "New SUITES definition for <suite name> (but not installed)." prints in the prompt window. This message results from the way ROOMS interacts with the File Manager. For an explanation of the meaning of this message see Appendix A, Programmer's Guide to ROOMS, "Suites."

"Show Suite"

Selecting the command "Show Suite" opens the "Show" menu (Figure 7-11) listing all the known suites. After you select a suite from the "Show" menu, ROOMS prints a list all of the rooms in that suite in the prompt window.

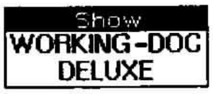


Figure 7-11. The "Show" menu

"Augment Suite"

Selecting the command "Augment Suite" allows you to add a room to a pre-existing suite. First the "Augment Suite" menu (Figure 7-12) opens listing all the known suites. ROOMS then opens a "Select" menu listing all the rooms that are not in any suite (except Pockets and Original neither of which can be added to a suite). The room selected is added to the selected suite.

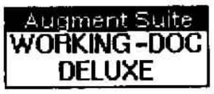


Figure 7-12. An "Augment Suite" menu

If all the known rooms are part of a suite ROOMS prints the message, "All rooms are already in some suite." in the prompt window and the operation is cancelled.

APPENDIX A: PROGRAMMER'S GUIDE TO ROOMS

This appendix contains the information needed to manipulate ROOMS programmatically.

All documented functions and variables are exported symbols in the package ROOMS which uses the packages `LISP` and `XCL`. The examples are printed as if the following had been evaluated:

```
(in-package "XCL-USER")
(shadow 'room)
(use-package "ROOMS")
```

Note: `(shadow 'room)` is evaluated to resolve conflict with the Common LISP function room. After this is done the Common LISP function room must be typed as cl:room. See *Common LISP the Language* by Guy Steele for an explanation of packages and shadowing symbols.

Rooms

In general, if you want to change a room programmatically you should proceed in the following way:

1. Call `update-placements` (see below).
2. Change the room programmatically.
3. Call `room-changed` (see below).

A room is a structure (see *Common LISP the Language* by Guy Steele, Chapter 19, "Structures") that, with one exception, behaves as if defined by:

```
(defstruct room
  (name nil :read-only t)  ; the name of the room as a string
  placements               ; list of placement objects
  inclusions               ; list of names of included rooms
  background               ; a background object
  tty-process              ; process to give TTY in this room
  props                    ; property list
  )
```

The exception is for the constructor function **`make-room`**, which has one mandatory argument, *name* (see below).

`(make-room` *name* `&key` *:placements :inclusions*
:background :tty-process `&allow-other-keys)` [Function]

Makes a room and adds it to the existing set of rooms.

name	the name of the room, typically a string. Room names are compared with `equal`.
:placements	a list of placements that go into the room. The items in this list are of the type returned by either **`make-placement`** and/or **`find-placement`**.

:inclusions a list of room names to be included in this room.

:background a background specification (see Chapter 3 Rooms "Edit a Room," and "Room Backgrounds," below for a description of a background specification.) If `nil`, the default light gray background is used and the name of the room is printed in the lower left-hand corner.

:tty-process specifies which process to give the tty to when the room is entered.

Examples of room creation:

```
(make-room "My Room")
```

Creates a room called "My Room."

The more complicated **make-room** example below works as long as you have an Exec window bound to **exec-window**.

```
(make-room "My Other Room"
  :inclusions '("Template")
  :placements (list (make-placement exec-window))
  :background '((:whole-screen 43605)
                (:region (0 0 1.0 1/4) :shade 33825)
                (:region (0 1/4 1.0 1/128) :shade
                         (:eval il:blackshade)))
  :tty-process (il:windowprop exec-window 'il:process))
```

Creates a room called "My Other Room" with the given background specification, includes the room "Template," and adds the placement exec-window. When you enter "My Other Room" the tty is given to exec-window.

current-room [Global Variable]

Value is the room that you are currently in. You should never set this variable.

(**room-named** *name*) [Macro]

Returns the room named *name* or `nil` if *name* doesn't exist.

For example,

```
(room-named "Mail") ⇒ #<Room "Mail">
```

`room-named` is especially useful for passing rooms to those functions that require a room as an argument, e.g., **room-prop**, `delete-room`, etc.

(**room-prop** *room prop* &optional *new-value*) [Macro]

Accesses the props field of *room*.

If *prop* is undefined for *room*, `room-prop` adds it to *room*'s property list.

If a *new-value* is supplied then the value of *prop* is changed to *new-value* and new-value is returned.

For example,

```
(setf (room-prop (room-named "Mail") 'my-property) 5)
```

changes the property **my-property** for the room named Mail to 5 such that:

```
(room-prop (room-named "Mail") 'my-property) ⇒ 5
```

(delete-room *room***)** [Macro]

Deletes *room*.

Note: **delete-room** deletes *room* from the environment even if it is part of a suite. This behavior differs from that exhibited when you try to delete a room that is part of a suite using either the ROOMS submenu or the Overview.

For example,

```
(delete-room (room-named "work room"))
```

deletes the room named "work room."

(rename-room *room new-name***)** [Function]

Copies *room* to new name and deletes it, effectively changing the name of *room* to *new-name*. Returns the new room created.

Note: You cannot actually change the name of a room, as this slot of the room structure is read-only.

(room-changed *room reason***)** [Function]

Informs the ROOMS system that *room* has changed. **room-changed** does some housekeeping on its own and then calls ***room-changed-functions*** (see below).

reason is one of:

:edited indicates that you have edited the definition of *room*. Especially important if you have changed the background or inclusions of *room*.

:placements indicates that the placements in *room* have been changed.

Note: If you have edited the room *and* changed the placements, giving *reason* a value of **:edited** is sufficient for ROOMS to notice all the changes.

room-changed-functions [Variable]

A list of functions that are called everytime you call (or the system calls) room-changed. Each function on the list is called with two arguments: the room that has changed and the reason for the change.

The reason passed here is one of:

:edited indicates that the room has been edited.

:placements indicates that the placements in the room have been changed.

:created indicates that the room has been newly created. Useful if you create a new room with the same name as an existing room.

:deleted indicates that the room has been deleted.

Note: The ROOMS system itself uses the **:created** and **:deleted** reasons, e.g., when you call **delete-room** the system calls **room-changed** with reason **:deleted**. However, the fact that **:created** and **:deleted** are reasons reserved by the system does not prevent you from having functions on ***room-changed-functions*** that are called when rooms are created and/or deleted.

pocket-room-name [Variable]

Value is a room name that is the current Pockets room. Default value is "Pockets". If you change ***pocket-room-name*** you must call **room-changed** (see above) on the new Pockets room so that the change is noticed by ROOMS. See Chapter 1, Introduction for information on Pockets.

Executing Code When Entering and Exiting Rooms

ROOMS provides facilities for calling functions, macros, etc. when you enter and exit rooms. Use the variables ***room-entry-functions*** and ***room-exit-functions*** to execute code when entering and leaving every room in the environment.

Use the room properties **:before-entry-functions** and **:before-exit-functions** to limit execution of code to specific rooms.

Use the macro **room-unwind-save** to make sure that code executed on room entry provides for some action on room exit.

You can put either named functions or the code itself on the variables and room properties .

room-entry-functions [Variable]

A list of functions that are called *before* a room is entered. The functions on the list are called with one argument—the room you are entering.

Note: If you also want to keep track of the room you are exiting, *current-room* is still bound to the room you are exiting when the room entry functions are called.

room-exit-functions [Variable]

A list of functions that are called *before* a room is left. The functions on the list are called with one argument—the room you are exiting.

:before-entry-functions [Room property]

A list of functions that are called when the room is entered. Each function is passed one argument, the room being entered. *current-room* may be used to find the room being exited.

This property is inherited i.e., the :before-entry-functions of included rooms are also called.

Note: The "before" in the name of this property is something of a misnomer. These functions (and functions on ***room-entry-functions***) are called just after the background has been painted but before any windows have been brought up.

`:before-exit-functions` [Room property]

A list of functions that are called before leaving the room. Each is passed one argument, the room being exited.

This property is inherited, i.e,, the `:before-exit-functions` of included rooms are also called.

Frequently, code that is executed on room entry will want to ensure some other computation is performed on room exit. To facilitate this use the following:

`(room-unwind-save &body` *body*`)` [Macro]

Causes body to be evaluated the next time a room is exited.

For example, a room entry function that allowed some rooms to have a directory that became the connected directory on entry could be defined as follows:

```
(defun cd-room-entry-function (room)
;;; If room has a :directory property, then connect
;;; to that directory, ensuring that the current
;;; connected directory will be restored on exit from room.
  (let ((new-directory (room-prop room :directory))
        (old-directory (il:directoryname t)))
    (when new-directory
      (room-unwind-save
        (il:cndir old-directory))
      (il:cndir new-directory))))
(pushnew 'cd-room-entry-function *room-entry-functions*)
```

Note: Room properties can be added to rooms from the room editor (SEdit) by adding a new property/value pair.

You could also make one or more rooms appear in reverse video and have the rest display in regular video. The following is an example of putting code directly on `:before-entry-functions` and using it in combination with `room-unwind-save` to restore a state when the room is exited.

```
;;;  When entering my-room change to reverse video
;;;  but change back to regular video when exiting
(pushnew '(lambda (room)
  (room-unwind-save (il:videocolor nil))
  (il:videocolor t))
  (room-prop (room-named
my-room):before-entry-functions)
     :test 'equal)
```

Inclusions

When you view a room you see it and its inclusions. The sequence of included rooms is called the *visible* rooms. This sequence is computed in a breadth-first, duplicate eliminating manner. The following table illustrates how this works:

Table A-1. How Inclusions Work

Room	Inclusions	Visible Rooms	Comment
A	none	A Pockets	Pockets implicitly included
B	A	B A Pockets	simple inclusion
C	B	C B A Pockets	indirect inclusion
D	B C	D B C A Pockets	breadth-first, duplicate elimination†

† The elimination of duplicates also serves to break inclusion loops.

`(do-inclusions (room-var` *room*`) &body` *body*`)` [Macro]

Evaluates *body* once for each room in *room*'s visible rooms with `room-var` bound to the visible room. A block named do-inclusions is created. Returns `nil`.

One might use something like the following code fragment to determine whether the room named "Test" was currently visible on the screen:

```
(let ((test-room (room-named "Test")))
  (do-inclusions (room *current-room*)
    (when (eq room test-room)
      (return-from do-inclusions t))))
```

You could make the :directory property shown in the `room-unwind-save` example above be inherited via the inclusion mechanism. You could replace the line `(room-prop room :directory)` with the following:

```
(do-inclusions (included room)
  (let ((directory (room-prop included :directory)))
    (when directory
      (return-from do-inclusions directory))))
```

This would cause inclusions to be searched for the `:directory` property.

Room Backgrounds

A background object is created using:

`(make-background` *external-form*`)` [Function]

external-form is similar to the value you give to the `:background` property when editing a room (see Chapter 3, Rooms, "Editing a Room"). `make-background` does error checking on the external form before returning a background object. You *cannot* use `make-background` to directly specify the value of `:background` in `make-room`.

The only time you need to use (and make) a background object is when you are accessing a room directly through the Common LISP structure mechanism. That is, you should use `make-background` only if you're setting the background field of a room structure. The `external-form` field of the

BACKGROUND structure is read-only. Thus, the idiom for changing backgrounds programmatically is:

```
(update-placements)
(setf (room-background <room>) (make-background <external-form>))
(room-changed <room> :editied)
```

`(background-external-form` *object*`)` [Function]

Used to access the external form of a room background. *object* is a background specification which can be accessed by:

```
(background-external-form (room-background <room>))
```

`*default-background-text-font*` [Variable]

The default font used for text printed on the background as defined by a room definition. To change `*default-background-text-font*` you must pass in a font descriptor. For example to change the font to Helvetica 12 you could:

```
(setq *default-background-text-font* (il:fontcreate '(il:helvetica 12
                                                        il:mrr)))
```

Initially background text is printed in 36 point TimesRomand bold.

You must call `internalize-all-backgrounds` (see below) before the background text change is propagated throughout the ROOMS system. The font of the current room's name is not changed until you exit and then reenter it.

`(internalize-all-backgrounds)` [Function]

Reprocesses background specifications.

Background specifications are processed before they're used to paint the background. This processing involves: (1) evaluating expressions within `:eval`; (2) converting fractional positions and regions to absolute positions and regions; and (3) caching bitmaps for background text. This processing does not take place every time a room is entered, but only when: (1) a room is created; (2) a room is edited; or (3) a sysout containing ROOMS is re-booted on a machine with a different sized screen.

While this catches most of the cases where background specifications need processing, there are some cases which this will miss, e.g., when a variable referenced within a `:eval` is reset; or when you change the value of the variable `*default-background-text-font*`. It is in these cases that one must explicitly call `internalize-all-backgrounds`.

`(externalize-region` *region*`)` [Function]

Translates *region* from integers to fractions. For example:

```
(externalize-region '(150 150 150 100))
 ⇒ (25/192 50/287 25/192 100/861)
```

For a definition of regions see the *Interlisp-D Reference Manual*, Chapter 27, Graphics Output Operations.

`(internalize-region` *region*`)` [Function]

Translates *region* from fractions or floating point numbers into integers. If *region* is given in integers then *region* is returned.

For a definition of regions see the *Interlisp-D Reference Manual*, Chapter 27, Graphics Output Operations.

`(externalize-position` *pos*`)` [Function]

Translates the position *pos* from integers to fractions.

For a definition of positions see the *Interlisp-D Reference Manual*, Chapter 27, Graphics Output Operations.

`(internalize-position` *pos*`)` [Function]

Translates the position *pos* from fractions or floating point numbers to integers. If *pos* is given in integers then *pos* is returned.

For a definition of positions see the *Interlisp-D Reference Manual*, Chapter 27, Graphics Output Operations.

Placements

Placements are not created for new windows until the room is changed. To force the creation and update of placements call `update-placements` (see below). This guarantees that every window on the screen has a placement.

A placement is a Common LISP structure that behaves as if defined by:

```
(defstruct placement
  window          ; the window to be placed
  region          ; the region to place it in
  shrunken?       ; true if the window is shrunken
  icon-position   ; position of the icon, if any
  props           ; property list
 )
```

The exception is for `make-placement` which is called with the single argument *window* and has no keyword arguments associated with it.

`(make-placement` *window*`)` [Function]

Makes and returns a placement for *window*.

`(update-placements)` [Function]

Updates the placements for the current room. `update-placements` reconciles the current room and its inclusions with what's on the screen; it assumes that what's on the screen is correct.

Placements are not updated every time one is modified because this requires a high overhead. Instead, placements within a room only are updated when the room is exited.

Therefore, in order to get the current state of any placement you may need to call `update-placements` before retrieving placement objects within a room.

(`placement-prop` *placement prop* `&optional` *new-value*) [Macro]

Accesses the props field of *placement.*

prop specifies the name of the property. If *prop* is undefined for *placement*, `placement-prop` adds it to *placement*'s property list.

If a *new-value* is supplied then the value of *prop* is changed to *new-value* and new-value is returned.

(`find-placement` *window* `&optional` *room*) [Function]

Returns the effective placement for *window* with respect to *room*, or `nil` if none exists. The effective placement is the placement that would be placed if *room* were entered.

Note: Before using `find-placement` you should call `update-placements` unless you are sure the placements are up-to-date.

window must be a main window, not an attached window or an icon.

room defaults to `*current-room*`.

(`all-windows &optional` *include-hidden?*) [Function]

Returns a list of all main windows (i.e., no attached windows are included) visible in the current room. If *include-hidden?* is true then all main windows in all rooms are included in the list.

(`lost-windows`) [Function]

Returns a list of all windows not in any room. If all windows are in some room returns `nil`.

Hidden Windows

ROOMS is implemented using *hidden* windows. Any window not in the current room is called hidden. Hidden windows are just like open windows except that they're not visible on the screen.

All the Interlisp-D window manipulation functions work on hidden windows. Also, all graphics operations work normally with hidden windows (See Chapter 27, Graphics Output Operations and Chapter 28, Windows and Menus in the *Interlisp-D Reference Manual.*)

(`hide-window` *window*) [Function]

Makes *window* hidden i.e., invisible yet open.

(`un-hide-window` *window*) [Function]

Makes *window* visible and open.

(`window-hidden?` *window*) [Function]

Returns `t` if *window* is hidden, `nil` otherwise.

Navigation

The navigation functions allow you to move between rooms.

(`go-to-room` *room* `&key` *:no-update :baggage*) [Function]

Spawns a process which moves you into *room* and returns the process handle. Control is immediately returned to the spawning process. `go-to-room` is usually called from menus and buttons, when the mouse has the keyboard. A process is spawned to allow the mouse to return the keyboard before rooms are changed. To synchronously change rooms, the following idiom is recommended:

```
(il:process.result (go-to-room room) t)
```

If *:no-update* is true then `update-placements` is not called before rooms are changed.

:baggage is a list of placements that are to be placed in *room*.

(`interactive-go-to-room-named` *name*) [Function]

Takes you to the room named *name*.

If the COPY or MOVE key is depressed at command execution, you are prompted to select those windows you want to take with you as baggage (see Chapter 5 Navigation, "Baggage" for more information).

This function is called by doors.

The Overview

These functions, used by the ROOMS Overview, are provided so that you can customize the appearance of the Overview in your environment.

(`get-pe` *room-name* `&optional` *region*) [Function]

Adds a placement editor window for the room named *room-name* to the current room. If *region* is specified then the editor will be shaped to occupy this region.

(`room-sort-function` *room-1 room-2*) [Function]

Used by the ROOMS system to sort rooms; you can redefine it to achieve other orderings. Determines the order of rooms in various menus and in the Overview. Should return true when *room-1* should appear before room-2. The default definition of this function orders rooms alphabetically by name.

It is possible to get into a very difficult state if you make an error when you redefine `room-sort-function`. In order to allow you to recover from such a state the original code is as follows:

```
(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)))))
```

`(reset-overview)` [Function]

Deletes old Overview and creates a new one. You should not call this function while in the Overview itself.

Buttons

Buttons are a unique user-interface device that provide for the execution of commands at the click of the mouse. Doors are buttons that move you from one room to another.

The functions and variables listed below deal with both doors and buttons.

Creating Buttons

`(make-button &key` *:type :text :text-form :action :help :font :shadows :inverted?*`)` [Function]

Returns a button. For example,

```
(make-button
  :text "CreateW"
  :action '(il:createw)
  :help "Creates a window")
```

Returns a button that, when given as the *button* argument to `make-button-window` (see below), creates a window with the word *CreateW* written on it. Selecting the window with the left-mouse button calls `il:createw`; holding the mouse button down for a specified period of time prints the message "*Creates a window*" in the prompt window.

:type specifies the type of button. Default is the value of `*default-button-type*` (initially `:shadowed`). Other types allowed are `:door`, `:stretchy-ark`, `:stretchy-round-ark` `:ark`, `:round-ark`, `:porthole`, and `:transparent`. These

standard button types are pictured in Chapter 5 Navigation, "Creating Doors."

Note: You can define your own button types. See **`def-button-type`**, below.

:text — specifies the text to be written on the button. *:text* can be either a string or a single-word atom (with no escape delimiters). For example, either "Snap Shot" or SNAPSHOT is allowed.

:text-form — used to compute the text written on a button and is `eval`'d every time the button window is redisplayed. *:text-form* allows you to have a button whose text changes. If *:text-form* is a list it is passed to `eval`. Otherwise the system assumes the value of *:text-form* is a function that is funcalled with the button window as its single argument. In both cases the result of the computation should return a string that is to be the text of the button.

Note: You should generally use either *:text* or *:text-form* not both. If you don't want the text written on the button to change then use *:text*. Similarly, if you want the text to change use *:text-form*

:action — specifies the action to be taken when the button is selected with either the left or middle button. If *:action* is a list it is passed to `eval`. Otherwise the system assumes the value of *:action* is a function and calls it with the button window as its single argument.

:help — supplies the help string that is printed in the prompt window when the left-mouse button is held down.

:font — supplies the font of the text written on the button. Should be given in the form (FAMILY SIZE FACE); for example, (MODERN 12 MRR).

:shadows — specifies the shadow specification for this button's text. Typically `t`, `nil`, a named shadow specification, or a shadow specification. Text shadow specifications are described in detail in "Text Shadows" later in this chapter.

If this argument is not provided to `make-button` then the button type is examined for default shadows (see `def-text-shadows`). If none are found in the button type then the value of the variable `*default-button-shadows*` is used.

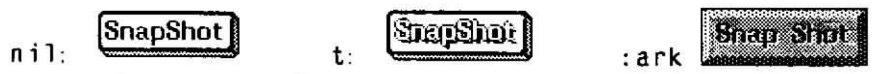


Figure A-1. Different button looks for different values of :shadows

Note: You can specify more complex shadowing. See "Text Shadows," **`def-text-shadows`** below.

:inverted? — if true, specifies that the button should be inverted. A `nil` value means that the button should be regular (Figure A-2). The default value is `nil` .

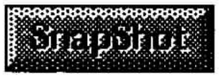
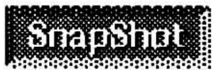
`nil:`  `t:` 

Figure A-2. Examples of regular and inverted buttons

`*default-button-shadows*` [Variable]

Used by **`make-button`** when *:shadows* are not specified and *:type* has no default shadows. Default value is `nil`.

`(make-button-window` *button* **`&optional`** *position***`)`** [Function]

Returns a window containing *button*. Create *button* using **`make-button`**.

position lets you specify the position where the window is placed using the dotted pair notation (*left* . *bottom*).

`(button-prop` *button prop* **`&optional`** *new-value***`)`** [Macro]

Accesses the props field of *button*.

If *prop* is undefined for *button*, **`button-prop`** adds it to *button*'s property list.

If *new-value* is supplied then the value of *prop* is changed to *new-value* and *new-value* is returned.

Defining Button Types

`(def-button-type` *name* **`&key`** *:image :mask :margins :default-shadows***`)`** [Definer]

Defines a button type and adds the type to the "Select Button Type" menu.

name	the name of the button type, typically a keyword. Used to refer to a button type when making and editing buttons.
:image	the bitmap to use as the button image.
:mask	if provided, it is a bitmap that determines which parts of *:image* to use. This allows non-rectangular images.

:image and *:mask* can be ordinary bitmaps, east-west bitmaps, north-south bitmaps, or nsew bitmaps (descriptions of these follow). The names of the bitmaps are derived from their relative positions as if they were distibuted around a compass. Both *:image* and *:mask* should be the same type of bitmap with the same dimensions.

:margins	if specified, is a list of four integers (left bottom right top) indicating the margins within which the text should be centered.
:default-shadows	if specified determines the default shadows for buttons of this type (see `make-button`)

A simple def-button-type example:

```
;;; A button type for 100 × 100 white buttons.
(def-button-type :boring
   :image #.(il:bitmapcreate 100 100))
```

Making Stretchy Buttons

A button type is called "stretchy" when *:image* and *:mask* are east-west bitmaps, north-south bitmaps or nsew bitmaps (see below for definitions of these types of bitmaps). Buttons whose *:type* is stretchy resize themselves to fit their *:text*—hence the term stretchy. All of the pre-defined stretchy buttons use nsew-bitmaps for their *:image* and *:mask.*

`(make-east-west-bitmap &key` *:east :center :west*`)` [Function]

Makes an east-west bitmap. *:east, :center,* and *:west* should all be ordinary bitmaps of the same height. When an east-west bitmap is displayed the :east bitmap appears on the right, the :center bitmap is repeated one or more times across the middle, and the :west bitmap appears on the left (Figure A-3).

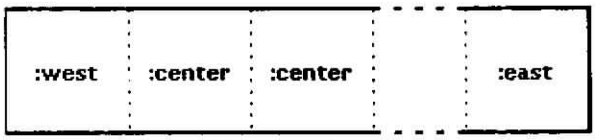


Figure A-3. How east-west bitmaps are stretched

`(make-north-south-bitmap &key` *:north :center :south*`)` [Function]

Makes a north-south bitmap. *:north, :center* and *:south* should all be ordinary bitmaps of the same width. When a north-south bitmap is displayed the :north bitmap appears on the top, the :center bitmap is repeated one or more times down the middle, and the :south bitmap appears on the bottom.(Figure A-4).

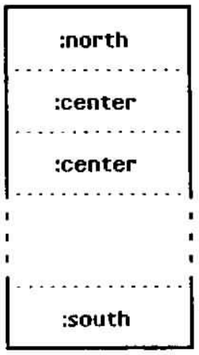


Figure A-4. How north-south bitmaps are stretched

`(make-nsew-bitmap &key` *:nw :north :ne :east :center :west :sw :south :se*`)` [Function]

Makes an nsew bitmap. All arguments are ordinary bitmaps with the following constraints on their size:

:nw, :north and *:ne* should all be the same height.

:east :center and *:west* should all be the same height.

:sw, :south, and *:se* should all be the same height.

:nw, :east, and *:sw* should all be the same width.

:north, :center, and *:south* should all be the same width.

:ne, :west and *:se* should all be the same width.

When a nsew bitmap is displayed the *:nw*, *:ne*, *:sw*, and *:se* bitmaps are placed in the corners. the *:west* bitmap is tiled down the left, the *:east* bitmap is tiled down the right, the *:north* bitmap is tiled across the top, the *:south* bitmap is tiled across the bottom and the *:center* bitmap is used to fill in the middle (Figure A-5).

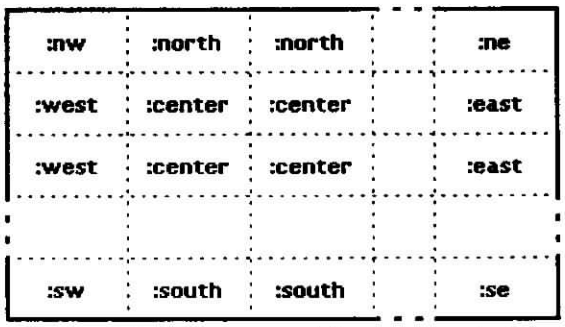


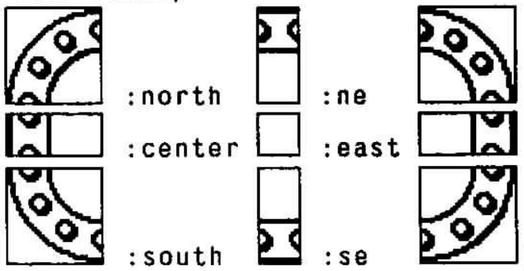
Figure A-5. How nsew bitmaps are stretched

Examples of Using def-button-type

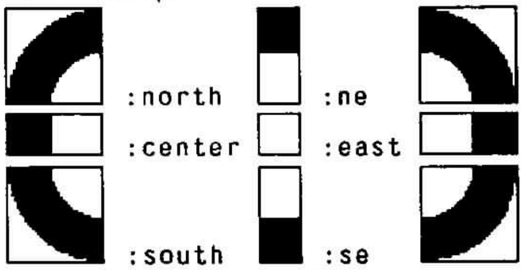
```
;;; A button type for stretchy white buttons
(def-button-type :stretchy-boring
   :image #.(make-east-west-bitmap
                        ;; all bitmaps must have same height, but
                        ;; may have different widths
                         :east (il:bitmapcreate 2 20)
                         :center (il:bitmapcreate 3 20)
                         :west (il:bitmapcreate 4 20)))
```

Here's an example you can't type in because you need to define the various bitmaps and masks. This example shows how the ROOMS porthole button is laid out to be stretchy and round.

```
(def-button-type :porthole
   :image #.(make-nsew-bitmap
                      :nw          :north       :ne
                      :west        :center      :east
                      :sw          :south       :se          )
```



```
   :mask #.(make-nsew-bitmap
                      :nw          :north       :ne
                      :west        :center      :east
                      :sw          :south       :se          )
```



```
   ;; indent all the way around as we want text to appear
   ;;over the hole, not over the bolts
   :margins (15 15 15 15)
   ;; as the text is over the hole, we need shadows to
   ;; make sure that it will be legible over any background
   :default-shadows t)
```

The inspector provides a handy way to edit east-west, north-south and nsew bitmaps. Thus, you might develop a new button type as follows:

```
;;; first make a blank east-west bitmap
(setq temp
  (make-east-west-bitmap
    :east (il:bitmapcreate 6 15)
    :center (il:bitmapcreate 4 15)
    :west (il:bitmapcreate 6 15)))

;;; then edit its contents from the inspector
(inspect temp)

;;; finally define the button type and make a button to test it
(def-button-type :my-type :image #.temp)
(make-button-window
  (make-button
    :text "testing..."
    :type :my-type))
```

Additional Button Variables and Functions

`*default-text-font*` [Parameter]

The default font of the text to be written on doors and buttons. Initially, 12 point Helvetica bold; value must be a font descriptor.

`*default-button-type*` [Parameter]

Default looks type for buttons and doors. One of `:shadowed`, `:door`, `:porthole` or `:transparent` or a user-defined button type (see `def-button-type`, above). Initial value is `:shadowed`.

`*button-help-delay*` [Global Var]

The period of time that a mouse button has to be depressed on a button or door before the help string is printed in the prompt window. Initial value is 1200 milliseconds.

`*button-selection-shade*` [Parameter]

Shade covering button or door to indicate that the mouse has selected it. Initial value is 32768. If you set `*button-selection-shade*` to `il:blackshade` then buttons are inverted.

(`set-button-window-text-string` *window string*) [Function]

Replaces the text in the button window *window* and reshapes the window if necessary i.e., everything required but redisplay. Typically followed by a call to `il:redisplayw`.

(`with-button` *action text help*) [Function]

Lets you create a button by holding down the COPY key while making a menu selection.

action	the action to be taken when the button is selected with either the left or middle mouse button.
text	the text to be written on the button.
help	help string that is printed in the prompt window.

For example, if you hold down the COPY key while selecting "Go to Room" from the ROOMS submenu, a button is created (Figure A-6) instead of the function selected being executed.

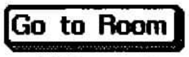


Figure A-6. A "Go to Room" button

To use **with-button**, change your code where you specify the action for a selected menu item to a call to **with-button**. The menu action is specified in *action*.

For example, create a menu (Figure A-7) with the following code:

```
(il:addmenu
  (il:create il:menu
    il:items il:_
     '(("Snap Shot"
        (with-button
          '(il:snapw)
          "Snap Shot"
          "Take a snap shot of screen"))
       ("VMEM Size"
         (with-button
          '(format il:promptwindow
                             "~%VMem  size  is  ~D~%"
(il:vmemsize))
          "VMEM Size"
          "Print VMEM size")))
    il:title il:_ "Sample Menu"))
```

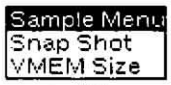


Figure A-7. Menu created incorporating with-button

Holding down the COPY key while selecting "VMEM Size" creates a button (Figure A-8).

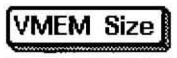


Figure A-8. A VMEMSIZE button

Selecting that button prints the current VMEMSIZE to the prompt window.

(make-bio *button*) [Function]

Returns a button image object (see the *Interlisp-D Reference Manual*, Chapter 27, "27.16 Image Objects" for a discussion of the image object data type). Create *button* using make-button.

You can also create button image objects directly from buttons, see Chapter 5 Navigation, "Creating and Using Button Image Objects."

Suites

`*suite-directories*` [Variable]

Value is the search path used for the load suite command. Initial value points to the connected directory.

`*suite-file-type*` [Variable]

Contains the file extension for suite files. Initial value is the string `"SUITE"`.

For example, saving a suite named test would result in a file called test.suite being written to the connected directory.

Suites and the File Manager

ROOMS creates a new File Manager type called `il:suites`. Updating a suite redefines it by calling `il:putdef` (see the *Interlisp-D Reference Manual*, Chapter 17, File Package, "17.8 File Package Types") with `il:dfnflg` bound to `il:prop` (see the *Interlisp-D Reference Manual*, Chapter 10, Function Definition, Manipulation and Evaluation, "10.2 Defining Functions.") Thus, updating a suite is the same as loading a file `il:prop` (which is why you get the message "New SUITES definition for <name> (but not installed).")

Resetting ROOMS

`(reset)` [Function]

Resets the ROOMS environment to look as if you had just loaded the ROOMS software for the first time. Deletes all rooms. Sets up two rooms; Original, containing all the windows that you had before `reset` was called, and Pockets, containing only a prompt window. Also sets `*pocket-room-name*` to `"Pockets"`.

This function should be used sparingly since recovery means recreating your entire ROOMS environment.

Text Shadows

In both of the places where text can be specified (backgrounds and buttons) there is a `:shadows` option that generally has a value of `t` or `nil`. In fact this option can be one of:

`nil`	no shadows
`t`	shadows proportional to font
a symbol	name of defined text shadows (see `def-text-shadows`)
a list	explicitly specified shadows

To explicitly specify shadows, use a list of shadow specifications. Each is a property list specifying a call to `il:bitblt`. When displaying text, ROOMS makes a bitmap of the text and then calls `il:bitblt` once for each shadow specification in the specified order.

Thus a shadow specification is a list and can contain the following keywords:

`:dx`	the x offset for the blt; default is 0
`:dy`	the y offset for the blt; default is 0
`:operation`	the operation for the blt; default is `il:paint`, but can also be one of: `il:replace` `il:erase` or `il:invert`.
`:source-type`	the source type for the blt; the default is `il:input` but can also be one of `il:invert` `il:texture` or `il:merge`.
`:texture`	the texture for the blt; default is 0.

`(def-text-shadows` *name* `&rest` *shadow-specifications*`)` [Definer]

Defines a named shadow specification.

Rooms has the following pre-defined shadows:

```
(def-text-shadows nil ())
(def-text-shadows :ark
   ;; indented shadows, like those in ark buttons
   (:operation :erase)
   (:dx -1 :dy 1))
```

Examples

These examples show you the basic shadow specifications, then illustrate what the button looks like with the new shadow specifications.

Start with the standard "Go to Room" button produced from the ROOMS submenu (Figure A-9). The button is created when you select "Go to Room" from the ROOMS submenu with the COPY key held down.

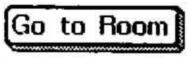
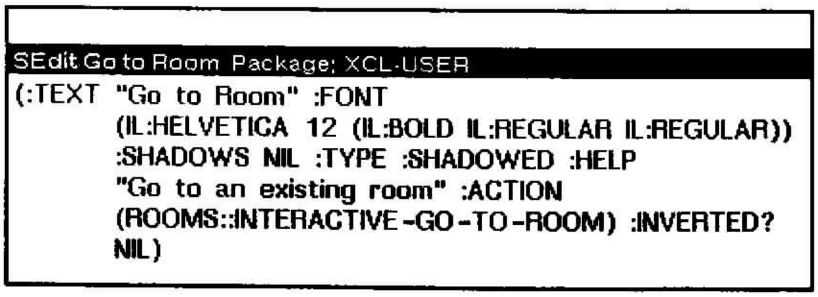


Figure A-9. A "Go to Room" button

You edit the "Go to Room" button by selecting the button, using the middle mouse button, while holding down the EDIT key. An SEdit window opens (Figure A-10).



SEdit Go to Room Package: XCL-USER

```
(:TEXT "Go to Room" :FONT
       (IL:HELVETICA 12 (IL:BOLD IL:REGULAR IL:REGULAR))
       :SHADOWS NIL :TYPE :SHADOWED :HELP
       "Go to an existing room" :ACTION
       (ROOMS::INTERACTIVE-GO-TO-ROOM) :INVERTED?
       NIL)
```

Figure A-10. The description of the "Go to Room" button

The following specifications change the shadowing of the "Go to Room" button (Figure A-9) and are entered after `:shadows` (Figure A-10).

Note: Comments are for illustrative purposes only and are not allowed in shadow specifications.

For example, to specify shadows like those **t** gives you, one might use:

```
((:dx 3 :dy -3)          ; print black shadow first, down and to right
 ;; print one bit off in each direction for black outline
 (:dx -1) (:dy -1) (:dx 1) (:dy 1)
 (:operation il:erase)); finally erase middle
```

Note: It is impossible to really specify shadows like those **t** gives you. **t** is magical in that it computes the shadows to look good with the font. Of course for a given font one could specify shadows which looked identical to those that **t** gives.

These new specifications produce a button like the one shown in Figure A-11.

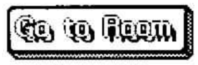


Figure A-11. The "Go to Room" button created with the new shadow specifications

An interesting variant is:

```
((:dx 3 :dy -3)          ; print black shadow first
 ;; print one bit off in each direction for black outline
 (:dx -1) (:dy -1) (:dx 1) (:dy 1)
 ;; paint middle gray
 (:operation il:invert :source-type il:merge :texture
          42405))
```

Which would produce a button that looks like the illustration in Figure A-12.

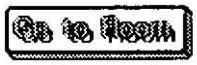


Figure A-12. A variation on the shadowed button

Window Types

When a room is saved onto a file as part of a suite, ROOMS goes through every window in the room and attempts to save it on the file. For a window to be saved, however, it must be one of a set of previously defined window types.

A window type definition gives ROOMS information about that window type so that ROOMS knows how to save it onto the file and retrieve it, saving enough information so that it can recreate that window at any time. ROOMS comes with window type definitions for many XAIE window types including:

- Prompt
- Exec
- TEdit
- Logo
- FileBrowser
- Chat
- Spy Button

Window Type Definitions

```
(def-window-type name &key :dependencies :recognizer :title
                           :abstracter :reconstituter
                           :placer :updater :files &allow-other-keys)    [Definer]
```

name is the name of the window-type, typically a symbol in the `keyword` package.

Each of the arguments, *:dependencies, :recognizer, :title, :abstracter, :reconstituter, :placer,* and *:updater* have a value of either a lambda expression or a function call (except for *:title*, whose value can also be a string).

You can add other keyword properties to the window type; these are not used by ROOMS and are treated as a user-defined property list for that window type, accessible at any time.

:recognizer takes a window and returns true if the window is of the type being defined. *:recognizer* can be considered a membership predicate and is a mandatory part of a window type definition.

:abstracter takes a window and returns a window description suitable for saving onto a suite file.

:reconstituter takes the *:abstracter* output from the file and returns a pointer to that window. Used when loading a suite file.

:title determines how placements for windows of this type appear in pictograms (as in the Overview, for example). Usually a string but can be a function. There are three relevant cases:

1. If *:title* is a string and the placement is not shrunken then *:title* is printed as a title bar, if it fits.
2. If *:title* is a string and the placement is shrunken then *:title* is printed centered, if there is room.
3. If *:title* is anything other than a string then *:title* is funcalled with arguments of: placement, region and dsp. Placement is the placement in question; region is the area to display the placement within; and dsp is the displaystream to display the placement in.

:dependencies a list of other window types on which this window type is dependent. Tells ROOMS to ignore conflicts for these types. Thus *:dependencies* allows you to create specialized window types that are subsets of a more general case. For example, a TEdit window is a subset of textstream windows. *:dependencies* doesn't allow you to inherit other window type methods; you

must still define *:recognizer, :abstracter, :reconstituter,* etc. for the specialized window-type.

:placer a function called on a window before you enter a room that contains a placement for that window. Allows the window to have different appearance properties in the different rooms it appears in. Works with *:updater.*

:updater a function called on a window when you leave a room that contains a placement for that window. Allows the window to have different appearance properties in the different rooms it appears in. Works with *:placer.*

:files a list of files that must be loaded before any attempt is made to reconstitute windows of type *name*. For example, if the window type `:my-type` is contained on my-window-types and windows of `:my-type` are created by the module my-module, then the following window type definition causes my-window-types and my-module to be autoloaded when a suite containing windows of `:my-type` is loaded:

```
(rooms:def-window-type :my-type
  . . .
 :files (my-window-types my-module)
  . . .
  )
```

Most window type definitions need to use only *:recognizer, :abstracter, :reconstituter,* and *:title.*

You do not have to specifically tell ROOMS that a window is of a certain window type; a window is run through the *:recognizer* portion of each window type definition until a match is found. If no match is found, then the window is not saved to the suite file, the window is brought to the current room, flashed several times, and a message is printed in the prompt window.

`(print-pep-title-string` *string region dsp*
`&key` *:font :no-title-bar?*) [Function]

Prints *string* within *region* on *dsp* if it fits. if *:no-title-bar?* is specified, then *string* prints in the center of *region,* otherwise it is displayed as a title bar. Use *:font* to specify the font the string is displayed in, if it fits.

Debugging Window Types

ROOMS provides a simple facility for debugging window types. Its use is limited to determining if the recognizer works and if dependencies are specified correctly.

`(window-type` *window* `&optional` *no-error?)* [Function]

Returns the window type, if one exists, of *window*. If *window* is not a recognized type and *no-error?* is `nil` then `window-type` signals an error. If *no-error?* is `t` and *window* is not a recognized type or there is a conflict, `window-type` returns `nil`.

If **`window-type`** signals an error informing you that it can't find a type for *window* and you expect there to be such a type then *window*'s recognizer is faulty in some way.

If **`window-type`** signals a "type-conflict" error then you have defined window types that conflict. If you intended this conflict then you need to specify the type of one in *:dependencies* of the other.

Using il:deldef to Delete Window Types

If you want to eliminate a window type use **`il:deldef`**. Window types are File Manager types. So if you created a window type named :test you could delete it by calling:

```
(il:deldef :test :window-types)
```

Examples of Window Type Definitions

Included in the ROOMS software package are the unsupported Rooms User's modules. One module, called random-window-types, includes prettyprinted listings of how each window type is defined as well as the normal user documentation.

Miscellaneous

Note Windows

The ROOMS Introduction suite uses a new window type called a note window. This is simply a window that holds text and can be used as reminders, explanations, notes, etc. Examples of note windows and their menu interface are given below.

Note: If you don't have the ROOMS Introduction loaded you must load rooms-notes.dfasl.

(**`make-note-window &key`** *:region :title :string :font :read-only?*) [Function]

Makes a note window.

:region the region on the screen that the window will fill. If `nil` the system prompts you to sweep out a region.

:title the title of the note window. Defaults to "Note:".

:string the string to be contained in the note window. Defaults to " ". However, there is a menu interface explained below that can also be used to enter the text. The string must be terminated with a return.

:font the font of the text in the window. Default is the value of **`*default-note-window-font*`**. It is not possible to have multiple fonts in one note window.

:read-only? specifies whether or not the text is editable after the note window has been created. Default is `nil`; `t` means that the text is read-only.

`*default-note-window-font*` [Variable]

The default font of the text written in the note window. Initial value is the value of `il:boldfont` (10 point Helvetica bold).

Note Window Menu Interface

Note windows have a menu interface. Middle mouse button on the title bar of a note window opens a menu (Figure A-13).

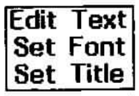


Figure A-13. The Note window top level menu

Edit Text allows you to enter and edit text.

Set Font opens up an SEdit window on the current font of the note window. Text in a note window can only be in one font so changing the font description changes all the text to the new font.

Set Title allows you to change the title of a note window. Prompts for the new title in the prompt window.

If you are in edit mode, middle mouse button on the title bar of the note window gives you another menu (Figure A-14).

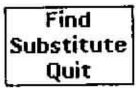


Figure A-14. The note window edit mode menu

Find allows you to search forward from the position of the caret for text. Enter the search string in the prompt window and press <RETURN>. If found, the text is underlined.

Substitute allows you to substitute one string for another in the *selected* text. In the prompt window enter the search string, the replace string, and whether or not you want to confirm each substitution.

Quit ends the text editing mode and switches menu control to the top level menu.

Note Window Example

```
(make-note-window :title "ROOMS Introduction")
```

Prompts for a region on the screen and creates an empty note window (Figure A-15).

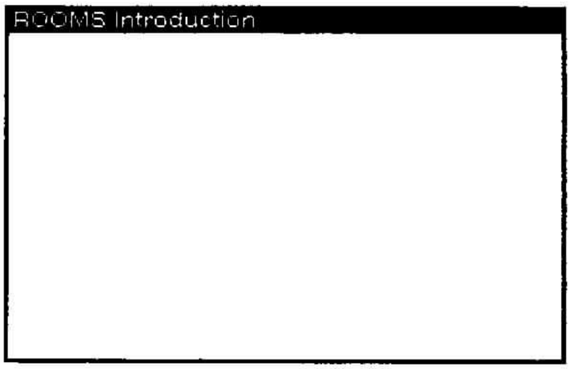
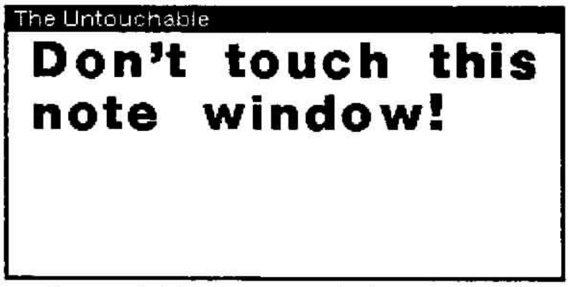


Figure A-15. A note window

```
(make-note-window :title "The Untouchable" :string "Don't touch this
          note window!" :read-only? t :font '(helveticad 24 mrr) )
```

Prompts for a region and creates a note window with text in it (Figure A-16).



Don't touch this note window!

Figure A-16. A note window containing some text

Since the note window is read-only, the note window menus are inactive.

Enhancing WHO-LINE

You can add a room entry to the LispUsers module Who-Line:

```
(pushnew *who-line-entry* il:*who-line-entries* :test #'equal :key
                              #'car)
(il:install-who-line-options)
```

[This page intentionally left blank]

GLOSSARY

baggage	placements copied when changing rooms
background	the shade, pattern, or bitmap that is displayed behind all the windows on the screen.
button	windows that allow you to execute commands at the click of a mouse. Doors are buttons that take you to another room.
current room	the room displayed on the screen; the room you are in.
destination room	a room that has other rooms included in it.
door	a specialized button that allows you to move between rooms
effective placement	the placement that is placed when room is entered.
hidden window	a window not visible on the screen
included room	a room whose placements are included in another room
inheritance	the way ROOMS establishes precedence of appearance when more than one room is included in another.
pe	placement editor
pep	placement editor placement
placement	structures that maintain information on a window (in the form of a pointer) and on a window's location in a particular room
Pockets	a special room whose placements appear in the same location and with the same appearance in every other room.
position	a point on the screen; given in x-y coordinates, as (x . y). If referring to the position of a placement, then it is the point on the screen that defines where the lower left-hand corner of the placement lies.
region	an area of the screen given in pixels; defined by (left bottom width height)
room	a virtual addition to the screen. Each room can be considered another screen.
ROOMS submenu	the menu off the "Rooms" entry on the background menu.
source room	a room included in other room(s)
stretchy button	a button whose region expands (or shrinks) depending on the extent of the text to be displayed in the button window.
suite	a collection of rooms that can be saved to a file and reloaded at a later time, restoring the environment (with regards to the rooms in the suite) to the same state as when the suite was written out.
virtual workspace	another name for a room.
window	The main window plus any attached windows and any associated icon. ROOMS treats all these as a unit.

[This page intentionally left blank]

Envos Corporation

READER COMMENT FORM

WE WOULD APPRECIATE YOUR COMMENTS AND SUGGESTIONS FOR IMPROVING THIS PUBLICATION.			
PUBLICATION NUMBER	RELEASE DATE	TITLE	CURRENT DATE

HOW DID YOU USE THIS PUBLICATION? ☐ LEARNING ☐ WRITING/GRAPHIC ☐ REFERENCE ☐ INSTALLATION	IS THE MATERIAL PRESENTED IN THIS GUIDE: ☐ FULLY COVERED ☐ WELL ILLUSTRATED ☐ WELL ORGANIZED ☐ CLEAR
WHAT IS YOUR OVERALL RATING OF THIS PUBLICATION? ☐ VERY GOOD ☐ FAIR ☐ VERY POOR ☐ GOOD ☐ POOR	DO YOU HAVE SUGGESTED CONTENT CORRECTIONS, CHANGES OR ADDITIONS? ☐ YES ☐ NO

YOUR OTHER COMMENTS MAY BE ENTERED HERE. PLEASE BE SPECIFIC AND GIVE PAGE, PARAGRAPH AND LINE NUMBER REFERENCES WHERE APPLICABLE.

YOUR NAME & RETURN ADDRESS

CUT ON DOTTED LINE

THANK YOU FOR YOUR INTEREST (FOLD AND FASTEN AS SHOWN ON BACK AND MAIL)

TAPE HERE ONLY

FOLD

No Postage
Necessary
If Mailed
in the
United States

BUSINESS REPLY MAIL

First Class Permit No. 1744 Mountain View, California

Postage will be paid by Addressee

Envos Corporation
Attn: Customer Support
1157 San Antonio Road
Mountain View, California 94043

FOLD

en·vos