Envos Corporation
1157 San Antonio Road
Mountain View, California 94043
(415) 966-6200
California: (800) 824-6449
Continental U.S.: (800) 228-5325

September, 1988

Dear ROOMS User:

Attached is the documentation for the Rooms User's modules.

Note that most of these modules include source code. Thus besides being useful in themselves, these modules provide good examples of programmatic use of ROOMS.

Rooms User modules are user-contributed software, and as such are not supported Envos products. We merely distribute these modules in hope that they will prove useful.

Please direct questions, suggestions and problems with Rooms User's modules directly to their authors. We have included the ARPAnet addresses of the authors in the documentation for this purpose.

We encourage you to contribute more Rooms User's modules. The same policies which apply to Lisp User's modules apply here. For more information see the Lisp User's guidelines distributed with the Medley release.

Customer Support

ROOMS USER'S MODULES

300002
Medley Release
SEPTEMBER 1988

Address comments to:
ENVOS
User Documentation
1157 San Antonio Rd.
Mountain View, CA 94043
415-966-6200

ROOMS USER'S MODULES

PART NUMBER 300002

SEPTEMBER 1988

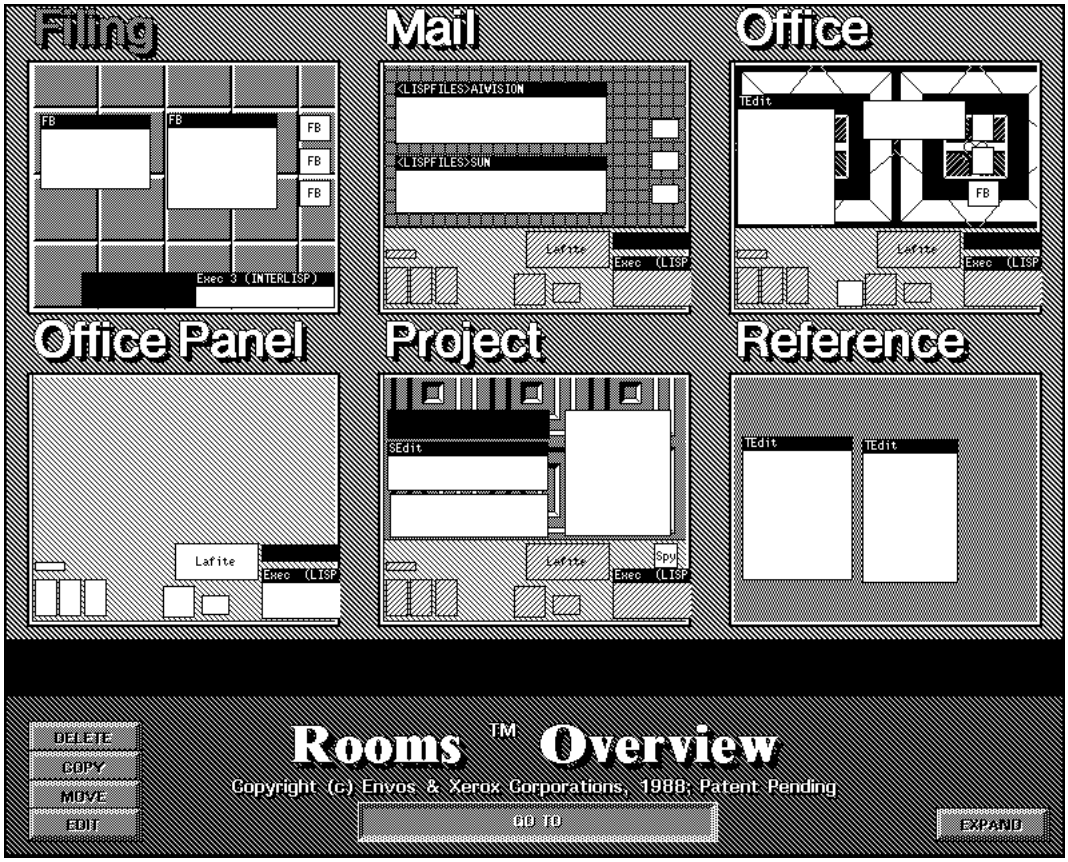
Text was written and produced with ENVOS text formatting tools; Xerox printers were used to produce text masters. The typeface is Optima.

Table of Contents

[This page intentionally left blank]

en·vōs

ROOMS™



Overview

ROOMS is a powerful interface to the Medley lisp environment window management system. ROOMS effectively increases the size of the screen and allows you to organize the work environment to facilitate management of complex parallel tasks.

For example a user may be developing and debugging a program, writing a paper, doing background research, reading mail, and filing all more or less at the same time. With limited screen space a user performing all these activities looses productivity shuffling through windows, icons, and other assorted screen clutter in order to reestablish the context of a task. ROOMS solves this problem of context switching by allowing users to create workspaces (called rooms), each one analagous to another screen, and containing only those tools needed for a specific task. Moreover, ROOMS provides methods for easily moving from one room to another and customizing the "look" of each room further aiding in efficient task-switching.

By providing dedicated workspaces, ease of navigation between workspaces, and a graphical link between each workspace and its related task, ROOMS helps users minimize context recovery time caused by task switching. In addition ROOMS provides an easy means to develop highly graphical custom interfaces for end-user applications. The overall result of this seamless integration of form and functionality is increased productivity in performing tasks in the Envos Software Development Environment.

Research Background

The ROOMS system design resulted from the following observations by members of the Xerox Palo Alto Research Labs:

1) Most intellectual work requires coordinating many sources of information, e.g. notes, drafts, spreadsheets, program listings, etc.

2) Before the widespread dissemination of desktop workstations and personal computers, knowledge workers typically spread paper containing these items out on a desk (or dining room table). But the small screens of computer workstations do not allow such a lavish use of space when working on many concurrent tasks. For example, it takes 22 average PC screens or 10 19" workstation screens to equal the area of a typical desk.

3) The result for overlapped windows systems is a kind of electronic messy desk, where the user spends large amounts of time moving, shrinking, and resizing windows in order to switch tasks.

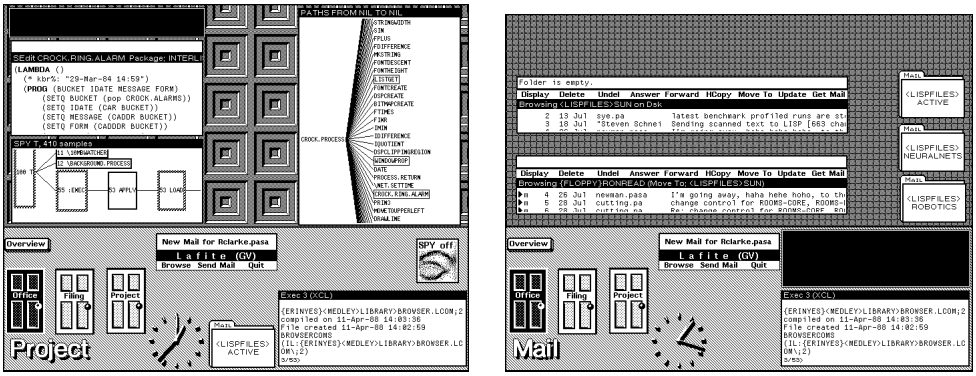
4) An interface designed to assist a user to work on concurrent tasks rather than assume linear work habits would have the following features:

   - Fast task switching and fast task resumption
   - Easy to re-acquire mental task context
   - Access to a large amount of information
   - Fast Access to information
   - Low user overhead
   - Engaged tools shareable among several tasks
   - Collections of engaged-tools shareable among tasks
   - Task-specific presentation of shared-engaged tools.

How Envos' implementation of ROOMS accomplishes these goals is described in the next section "The ROOMS Design".

The ROOMS Design

Rooms solves the problem of user task switching by allowing users to create a number of screen-sized workspaces called Rooms. The figure below shows two Rooms.



Two example Rooms (a) used for programming and (b) used for reading mail

In each room, there are a number of small icon-like objects called Doors. When a door is selected with the mouse, the user has the illusion of transiting to a new room containing windows. Each room is related to a different major task, such as working on a particular project, reading mail or file management. In the room are a number of engaged tools related to the task. When a user re-enters a room it appears exactly as when it was left.

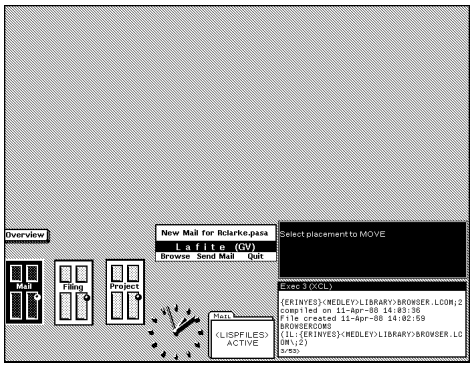
The basic notion of the Rooms scheme is simple. But in order for this basic notion to be successful the ROOMS system design has many features to help manage window interaction among rooms and to keep the user from getting lost.

Task Interaction

When working in ROOMS, there are usually a number of engaged tools that the user wants to share among different tasks. Examples of these might include a code editor, a text editor or a clock. The following ROOMS design features allow for the sharing of engaged tools:

Placements A window in a room has a specific location and shape. This is known as a Placement. The same window can exist in several rooms, with different locations, shapes, etc., in each room. Actions done on a shared window in one room are reflected in another room. Note that the black promptwindow in the Mail and Project rooms has a different location and shape in each room.

Inclusions Rooms can be included within other rooms. This allows collections of rooms to share a common set of windows, known as Inclusions. The band of windows and icons common in both parts of the Mail and Project rooms is a control panel, shown below, implemented as an included room.



Example of an included room called Office Panel

Baggage When moving between rooms you may carry windows with you. For example, you may wish to bring program code from one workspace to a workspace where you are doing documentation. This is accomplished by holding down the Move or Copy key when selecting a door. You will then be prompted to select the windows you wish to carry with you to the next room. The windows will have the same presentation in the new room.

Pockets You can also have a constant piece of baggage called a Pocket. A Pocket is a room dynamically included in all rooms. Whichever windows are placed in your Pocket (a clock say) will automatically occur (at the same location and presentation attributes) in all rooms.

Navigation

The organization of the user's workspace into a number of workspaces creates a potential navigational problem. ROOMS solves this problem with a number of navigational aids.

Doors Doors provide the basic link mechanism between rooms. Selecting a door with the mouse "moves" you to the new room containing windows associated with that room.

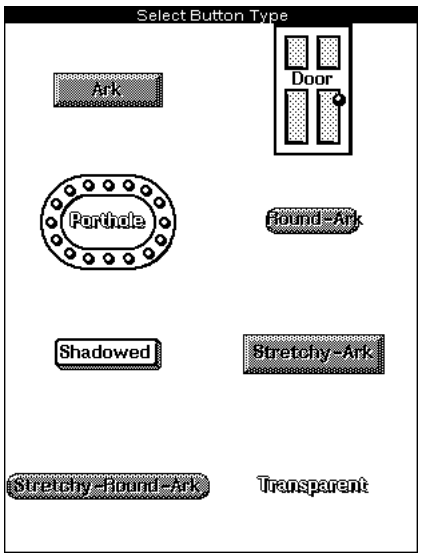
Back Doors Rooms allows you to create a Back Door in a room. If you have created a Back Door you can easily move back to the task you were previously working on before you entered the present room.

Rooms Menu Selecting the Go To Room sub-item from the ROOMS background menu presents you with a list of Room names. Selecting one will move you to that Room.

Overview The main feature of the Overview is a set of Room pictographs as shown in the openning diagram. From the Overview you can see the overall layout of a room and the tools it contains. The room pictographs can be instantly expanded one at a time, allowing you to browse through the windows in the entire set of rooms. In addition you can enter any room directly via the Overview.

Tailorability

Several mechanisms are provided to help users tailor rooms. First, simply creating, moving, deleting and shaping windows in the usual way causes things to exist in rooms. Thus ROOMS preserves the natural interaction with the user. Second, special background entries are provided to allow the user to create new doors and other conveniences of construction. Below is an example of the standard set of Doors available to users.



At the Overview level, it is possible to copy, move or delete window pictographs within a room and between rooms and have the changes reflected in the rooms themselves. And finally, Rooms has a simple layout language for creating unique backgrounds for Rooms. By using the structure editor, SEdit, on this layout language, users can run arbitrary procedures on the entrance and exit of a room and can compute

specialized backgrounds for rooms. Below is an example of a SEdit of a rooms layout expression for the Project room shown earlier.

```
SEdit Project Package: XCL-USER
(:INCLUSIONS ("Office Panel") :BACKGROUND
   ((:REGION (0 1/4 1.0 3/4) :SHADE
       (:EVAL ROOMS:RENAISSANCE-BITMAP)
       :BORDER 2)
    (:TEXT "Project" :FONT
           (IL:HELVETICA 36 IL:BOLD)
           :POSITION (10 . 10))))
```

Suites Having designed an environment to suit your needs, ROOMS allows you to restore that state if there is a need to reload your system. Subsets of rooms can be grouped together to form Suites. Suites can be saved to a file and later restored to a new enviroment or shared with other users. Most system windows have a ROOMS window-type specification which allows them to be saved and restored from a suite. Thus a Filebrowser window that you have open on a particular system can be saved and the window placement and contents will be restored when you reload the suite onto a new system.

Buttons Buttons are a unique user-interface device that provides for the execution of any lisp command at the click of the mouse. Buttons can be of any shape and size. Doors are examples of buttons that move you from one room to another. For example selecting the following button will perform a directory listing of the local disk.

The button definition is shown below:

```
SEdit Directory Package: XCL-USER
(:TEXT "Directory" :FONT
       (IL:HELVETICA 12
                     (IL:BOLD IL:REGULAR
                              IL:REGULAR))
       :SHADOWS :ARK :TYPE :STRETCHY-ROUND-ARK
       :HELP "Bring up local disk directory"
       :ACTION (DIR "{dsk}<lispfiles>")
       :INVERTED? NIL)
```

Customization of Rooms

When used as the interface to an application, ROOMS provides you with a complete programmatic interface to allow application specific customization. This includes the ability to have applications create and switch rooms under programmatic control as well as enable users to design custom buttons and window-types.

References

A Multiple, Virtual-Workspace Interface to Support User Task Switching, Austin Henderson/Stuart Card, 1987 ACM CHI + GI Conference Proceedings on Human Factors in Computing Systems and Graphic Interfaces

For further information about ROOMS contact your Envos Marketing Representative, or call toll-free in the Continental United States 1-800-228-5325, or in California 1-800-824-6449.

Envos Corporation

1157 San Antonio Road

Mountain View, California 94043

(415) 966-6200

>>Module Name<<

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

>>Type INTERNAL here if the file is for Internal Use Only<<

This document last edited on >>DATE<<.

INTRODUCTION

>>This paragraph should be replaced by an overview of your module.<<

MODULE EXPLANATIONS

>>Functions, Variables, and Lisp Code Examples<<

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system. Variables look like functions, except that the word ''Variable,'' enclosed in **square brackets**, follows the variable name. Please note that these are the characters [], not the parentheses

```
This is an example of code. It is in 10-point Terminal font.
```

Function names, commands, file names, and the like are in 10-point modern.

Be sure to include the following information in any module explanations:

- any file dependencies
- definitions of all arguments
- module, function variable, etc. limitations
- a liberal number of examples for all functions, variables, etc.

LAFITE-WINDOW-TYPES

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on January 28, 1988.

INTRODUCTION

This module provides window types for Lafite windows. Lafite is the Interlisp-D mail program. Loading this module enables you to save Lafite windows in your suite files.

EXPLANATIONS

We provide window types for Lafite browsers and for the Lafite status window.

Note: we do not provide window types for message windows, as these are far more transient than the status window and browsers.

OFFICE.SUITE

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on January 28, 1988.

INTRODUCTION

This is a sample suite file.

Beginning Rooms users might load this, augment it as desired, and then use the "Dump Suite" command to save their own version of it.

This suite also gives some ideas about how one might use the facilities which Rooms provides.

INSTALLATION

Load the file OFFICE.SUITE. This file (like all suite files) will load ROOMS if it is not already loaded. It will then create the rooms in the OFFICE suite. Use the Overview to see these rooms.

RANDOM-WINDOW-TYPES

By: Doug Cutting (Cutting.PA@Xerox.COM),
Stan Lanning (Lanning.PA@Xerox.COM) and
Ramana Rao (Rao.PA@Xerox.COM)

This document last edited on April 21, 1988.

This module contains window type definitions for many LispUsers modules.

These definitions serve two purposes. Their primary purpose is to allow one to save windows created by these modules in suites. Secondarily they provide good examples of window type definitions for the programmer attempting to define his own.

Modules covered are:

WHO-LINE
CALENDAR
PRINTERMENU
CROCK
BICLOCK
ADDRESSBOOK
PHONE-DIRECTORY
GRID-ICONS

In addition, support is provided for the FILEWATCH LispUsers module. This works by adding a property to each room which notes whether FILEWATCH is on in that room. This property is not inherited, i.e. including a room in which FILEWATCH is does not turn FILEWATCH on.

A listing of RANDOM-WINDOW-TYPES can be found in Appendix A of this document.

BACKGROUND-MENU-BUTTONS

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on January 28, 1988.

INTRODUCTION

BACKGROUND-MENU-BUTTONS makes it easy for the Rooms user to make buttons which do the same things as entries on the background menu.

EXPLANATION

When this module is loaded a button labelled "Make Background Button" is placed on the screen. When this button is pressed a menu which looks just like the background menu is raised. Selecting an entry from this menu will create a button which does the same thing as this entry.

>>Module Name<<

By: >>Your Name<< (>>Your net address<<)

Uses: >>Other modules necessary to run this one<<

This document last edited on >>DATE<<.

INTRODUCTION

>>This paragraph should be replaced by an overview of your module.<<

MODULE EXPLANATIONS

>>Functions, Variables, and Lisp Code Examples<<

It is usual to first give the name of a function, then describe its purpose and each of its arguments. When the name of a function is first given, it is set off like this:

(IMAGEFNSCREATE *DISPLAYFN IMAGEBOXFN PUTFN GETFN COPYFN*) [Function]

The function name is in 10-point regular Modern, all caps. Arguments are in 10-point italic Modern, in all caps, mixed case, or lowercase, as they appear in the system. Variables look like functions, except that the word ''Variable,'' enclosed in **square brackets**, follows the variable name. Please note that these are the characters [], not the parentheses

```
This is an example of code. It is in 10-point Terminal font.
```

Function names, commands, file names, and the like are in 10-point modern.

Be sure to include the following information in any module explanations:

• any file dependencies

• definitions of all arguments

• module, function variable, etc. limitations

• a liberal number of examples for all functions, variables, etc.

ROOMS USERS' RULES

This document describes the rules and procedures for "Rooms Users" modules. This document is mainly for Rooms Users' module writers, but users should also understand the rules.

DEVELOPING A ROOMS USERS' MODULE

A Rooms Users' module is a useful program made available to the general Rooms community. Neither the author nor the custodian of Rooms Users' imputes any warranty of suitability or responsibility for errors.

Rooms Users' modules should be easily distinguishable from released library and Lisp Users' modules. In particular, this means that a Rooms Users' module may not have the same name as a Library or Lisp Users' module and should be visibly different. Rooms Users' modules derived from released software should be announced to the public only after communicating with the organization responsible for that released software.

Testing is important. If you make significant changes to a Rooms Users' module, enlist users at your site as alpha testers. A Rooms Users' module is not shoddy software; it is software made available outside the regular release channels.

ROOMS USERS' MODULE OWNERSHIP

A module submitted for Rooms Users' remains the "property" of the submitter. Others may not make changes, except for their own private use, without negotiating with the owner (who may already be making similar or incompatible changes).

As the owner of a module, you are not required to fix bugs, but if not, you must be willing to transfer ownership (permanently) to someone who volunteers to fix them. Ownership may pass back and forth among several people as long as they agree.

SUBMITTING ROOMS USERS' MODULES

If you are not an internal user, you should submit your new module to us through e-mail or on a floppy or tape. External users should make sure that they include all relevant information, such as documentation containing an e-mail or US mail address where he/she can be reached.

SUBMITTING FILES TO ROOMS USERS'

As with released software, it is important to submit not just the resulting product, but all the files needed to build and maintain a Rooms Users' module:

1. the file to load (.LCOM or .dfasl or .SUITE)
2. documentation describing it, following the formatting rules (see below)
3. a source file that can be released (optional)
4. data files, if needed

Modules submitted once are released once. Do not assume that a module submitted for one release will be automatically released in subsequent releases.

DOCUMENTATION

No modules will be released without documentation. Documentation can be as simple as a paragraph describing what the module does and how to use it, or it can be as extensive as a dozen-page user manual. All modules should have a file with a .TEDIT extension. Formatting should be done according to the rules outlined in the Rooms Users' Template, included on the Rooms Users floppy or tape as EASYTEMPLATE.TEDIT and also printed in this document. All users, external users included, should follow the Rooms Users' Template rules. If the documentation is large and formatting time consuming, you can also produce an interpress file (with the .ip extension), as well as submitting a .TEdit file. (Be sure to update the interpress file if you update the documentation!) Documentation should include the full address of the submitter.

COMPATIBILITY

Any submitted Rooms Users' files should be compilable in a "vanilla" Rooms environment. The file itself should load in any auxiliary modules when necessary.

Thanks for your cooperation.

TOUCHY BUTTONS

By: Ramana Rao (Rao.pa)

Uses: Rooms

This document last edited on January 22, 88.

INTRODUCTION

This Rooms Users Package provides a number of "touchy" buttons i.e. buttons that visually depend on the state of the world and change the state of the world when touched. Right now I provide three types of touchy buttons: includer, toggler, and once-only. You can make a touchy button by calling rooms::make-<touchy-type>. I will take suggestions for any others that people think may be useful since I'm trying to abstract the touchy technology.

BUTTON CONSTRUCTORS

(ROOMS::MAKE-INCLUDER *ROOM-NAME*) [Function]

Includers allow you to conveniently mixin or mixout rooms. For example, you can have "Notecards-Mixin" and "Programming-Mixin" Rooms and have includer buttons in your personal "Pockets." Then you can include these functionality traits whenever you need them no matter the room.

(ROOMS::MAKE-TOGGLER *VARIABLE-NAME*) [Function]

Toggle buttons allow you to toggle boolean variables. This should obviously be generalized to something that allows you to select or circulate through value settings.

(ROOMS::MAKE-ONCE-ONLY *FORM INITIAL-TEXT FINAL-TEXT*) [Function]

A once-only button evaluates a form exactly once in a sysout and then displays that it is exhausted. For Example:

```
(ROOMS::MAKE-ONCE-ONLY (IL:PROMPTPRINT "Hello, World") "Fire..."
"Exhausted")
```

UN-HIDE-TTY

By: Doug Cutting (Cutting.PA@Xerox.COM)

This document last edited on April 28, 1988.

ROOMS changes the window system such that open windows are not always visible. Some applications presume that open windows are visible. This module attempts to amend this situation.

Often when an application decides to start using a window which is not visible it gives that window the caret. CHAT is an example of such an application. When IL:CLOSECHATWINDOWFLG is NIL (the default) CHAT windows are left open after their connection is closed. If one leaves such a CHAT window in another room and then attempts to open a CHAT connection, CHAT will merrily re-use the hidden window. But as CHAT takes the caret when it opens a connection, we can identify that window and pull it into the current room.

(ROOMS:UN-HIDE-TTY) [Function]

If the window with the caret is not in the current room, it is brought into the current room with a call to ROOMS:UN-HIDE-WINDOW. If the window with the TTY is already visible on the screen then it is flashed. If there is no TTY window (i.e. the process with the keyboard has no TTY window) then a message to this effect is printed to the prompt window.

One can use this function in the action of a button, e.g.:

```
(ROOMS:MAKE-BUTTON-WINDOW
  (ROOMS:MAKE-BUTTON :TEXT "Un-Hide TTY" :ACTION '(ROOMS:UN-HIDE-TTY)))
```

Control-Y [Interrupt]

Brings the window with the caret into the current room by calling ROOMS:UN-HIDE-TTY. This interrupt is installed when UN-HIDE-TTY is loaded. This will not work with applications which have their own interrupt tables, e.g. TEdit and CHAT.

ROOMS USERS' MODULES

INTRODUCTION

ROOMS User's modules are written by users of Envos ROOMS. They are supported by the individuals who wrote them, not by Envos. Envos takes no responsibility for the reliability and maintenance of these modules.

Each module is documented, some in detail, others with only enough information to get the user started. At the top of each document, the name of the author is written, along with his electronic mail or U.S. mail address. Please contact the author with any problems you may have.

Note that most of these modules include source code. Thus besides being useful in themselves, these modules provide good examples of programmatic use of ROOMS.

This document contains explanations of each module, written by the author of each. It also contains an explanation of how to submit your own ROOMS User' module. Please read these instructions carefully if you wish submit a module.

WALLPAPER

By: Doug Cutting (Cutting.PA@Xerox.COM) and
Larry Masinter (Masinter.PA@Xerox.COM)

Uses: SCREENPAPER

This document last edited on August 8, 1988.

INTRODUCTION

This module provides an easy way to create distinctive backgrounds for your rooms.

All symbols described in this document are in the package ROOMS.

FUNCTIONS

(MAKE-WALLPAPER-WINDOW &OPTIONAL REGION) [Function]

Makes and returns a Wallpaper window. When the LEFT or MIDDLE mouse button is pressed over Wallpaper windows the user is asked to select a tile size, then a position for the tile. Positions are selected with the LEFT button. Each tile is displayed in the window. If the user presses the MIDDLE button then the background of the current room is changed to be the current tile. The user can abort this process at any time by pressing the RIGHT mouse button.

There is a window type definition for Wallpaper windows so they may be saved in Suites.

(HACK-BACKGROUND SHADE &OPTIONAL ROOM) [Function]

Changes the first background shade specified for ROOM to be SHADE. If ROOM does not paint the background then this function adds a command to the background specification for ROOM which paints the whole screen with SHADE. ROOM defaults to the current room.