A Linguistic Characterisation of Design in Text-Based Virtual Worlds

Anna Cicognani

Laurea in Architecture (Hons)

Supervisor

Dr. James Rutherford

Associate Supervisor

Prof. Mary Lou Maher

A thesis submitted in fulfilment

of the requirements for the degree of

Doctor of Philosophy

Department of Architectural and Design Science

Faculty of Architecture

University of Sydney

APPENDICES

Appendix A (Acronyms and Glossary) is included in the paper version of his thesis.

List of Appendices Enclosed in the CDROM

APPENDIX B. Basic MOO Commands

When accessing a MOO, a user finds a series of commands written to easily perform communication, navigation, and construction. However, commands in MOOs are continuously written and customised: users can write code to adjust input and output to their needs. The linguistic interface, the commands a user utilises to interact with the MOO software, are flexible and malleable: by renaming a command or an entity, a simpler use can be provided. Some VWs, modelled on specific environments (eg. a spaceship) have a distinct way of giving outputs. For example, a MUD can respond with the following:

```
>say What do you think I am doing here?
Creeper announces, “What do you think I am doing here?”
```

others would output:

```
Creeper says, “What do you think I am doing here?”
```

or

```
Creeper asks, “What do you think I am doing here?”
```

or

```
Creeper babbles, “What do you think I am doing here?”
```

To enter in more details without analysing the modalities of command issuing, I believe that an overview of basic commands which can be used in a MOO and some examples of their effect will be helpful to understand the environment. I will therefore explain some of the most common commands and their responses (outputs). The ‘>‘ stands for the user’s input.

Communication

- *say*, to say something to the others in the same room:

  ```
  >say a nice dress that one!
  Creeper says, “a nice dress that one!”
  ```

- *whisper*, a private communication command between two users in the same room:

  ```
  >whisper what r u doing here to sneep
  You whisper, “what r u doing here” to sneep-the-beep.
  ```

- *page*, a private communication command between two users in different rooms:

  ```
  >page sneep hey u around???
  Your message has been sent.
  ```

And the recipient will see:

```
You sense that Creeper is looking for you in Creeper's Lake.
It pages, "hey u around???"
```

- *think*, to show text in a particular way:

```
>think I guess I am awake
Creeper . o O ( I guess I am awake )
```

- *@shout*, sometimes restricted to administrators only, to say something to everyone connected at that time:

```
>@shout I want everybody here now!
You shout, "I want everybody here now!"
```

and others will see:

```
Creeper shouts, "I want everybody here now!"
```

- *@send*, to send an MOOmail message, in the classic Email fashion

Other commands, like "murmur," the same as "whisper," or "sign," to hold up a sign for everyone to see the message written on it, are customisations of some MOOs.

Action

This is a more complex group of commands. Each user, according to the experience, has a set of commands for actions; also, a particular room can allow certain events, and therefore will have added commands which can be used by all the people in that room.

Some basic ones are:

- *get*, to pick up something:

```
>get mixer
You take the mixer.
```

Others will see:

```
Creeper takes the mixer.
```

- *drop*, to drop something:

```
>drop mixer
You drop the mixer.
```

Others will see:

```
Creeper drops the mixer.
```

- *give*, to give something to someone:

```
>give mixer to sneep
You hand the mixer to sneep-the-beep.
```

Sneep-the-beep, another user, will see:

```
Creeper hands you the mixer.
```

Other more specialised commands, like *open*, *close*, *put*, *sit*, *stand*, *start*, *stop*, *hush*, *play*, *wake*, and many others, have a similar effect to what they would do in a real life situation. The general guideline for commands is, in fact, natural language. Commands are programmed in such a way that there is a intuitive correspondence between what a command would do, and its effects. For example, the command "open" would have the effect of opening something in the MOO, as the action of opening would be in real life. This observation, which may at first sound banal, is in fact important to understand how linguistic acts in a text-based VWs correspond to actions in real life (and not necessarily speech acts).

The performance of commands in text-based VWs is only one of the aspects, albeit not very well studied, of the complex nature of these environments. It is crucial, at this point, to recognise that in MOOs language is not simply a communicative medium, but has got a constructive nature, and a consistent Design potential.

Navigation

Navigating the MOO means moving from one room, or space partition, to another. It can be done in two ways: *walking*, when rooms are contiguous, or *teleporting*, when they are not. Both walking and teleporting can be disabled, when a room must remain private, closed to some users or everyone. To walk, users type the name of the exit, or the direction:

```
>offices
You go to the office area.
```

or

```
>stairs
You go up the stairs.
```

To port from one room to another, the command "@go <name or number of room>" is used:

```
@go offices
```

or

```
@go #184
```

both will port a user to the Office Area, which corresponds to number #184 in the MOO database.

Bridges and Charitos (1996) argue that “teleportation may simplify navigation but may also prove disorientating as it corresponds to gaps in the cognitive maps generated by operators.” The design of the environment does not have to respond to physical constraints, neither it has to prove logical connections among space partitions, since at any time a user can *jump* from one place to another. The contiguity and continuity constraints which are inevitable in a physical environment become irrelevant in a VW. However, navigation is a major component in the design of the structure of VWs: since the layout map is not bound by proximity, other rules must be found when approaching the world design. For example, a network model, where rooms are connected to others with a similar theme or task, or a container model, where a bigger container with a theme includes other sub-partitions of related theme (eg. an office building with not only other offices, but also services, reception, and so on).

APPENDIX C. MOO Descriptions of Areas and Entities

The following descriptions of entities (areas and things) have been found during visits in other MOOs. For each MOO visited, I report the address and a short summary of what kind of rooms and navigation have been encountered. Follow examples of area and entity descriptions. The copyright remains with the correspondent MOO, respective authors, or howelse specified by each MOO.

RiverMOO

(river.honors.indiana.edu:8888)

Room descriptions: function, with some attention to details. In general, descriptions try to give an impression of how the room feels like (smells, colours).

Navigation: cardinal points or simple intuitive names (exit)

SAMPLES

```
City Slum (#2032)
This building was once a tire manufacturer, but when the building was
gutted by fire in 1954, it was abandoned. You can still see the scorch-
marks on the walls.
Now, it has been made into a place where those of RiverCity's less
fortunate sort can sleep with a roof over their heads. But sometimes it
seems as though the roof is going to cave in. The walls are bare
concrete, with two barred, broken windows on the far corners of the
room.
There is a light fixture squarely placed in the middle of the ceiling.
It is swinging from a thin, fraying black cord. There are five cots
along each wall. The legs of the cots are rusted. The sheets of the
cots are gray and dirty.
The room is public. To make it your home, type @sethome.
There's a door leading back out to the Back Alley. The door is closed.
You are here.  Manischewitz (asleep), Hardwick (asleep), Muttman
(asleep), Translucent Adept (asleep), braindead  (asleep), shakes the
clown (asleep), and Curunir (asleep) are asleep here.
You see Slum Jack, Slum Phone, and BerryMan (asleep) here.

Obvious exit:  alley to Back Alley
ANSI Version 2.1 is currently active.  Type "?ansi-intro" for more
information.

Back Alley (#1785)
```

A dark, musty, back alley. The paving is still the old-fashioned cobblestone of an older generation. Its pathway is narrow, but accommodating for a midnight stroll. The buildings that line this alley are old, but they have been reasonably kept up. The doors are shorter than the average height of people nowadays, which adds to the piquancy of the place.
The colours of the buildings are the only things that set it a bit out of place. Once in a while, the odd building exhibits a bright colour, contrasting with the drab old shades of an age gone by. The place calls out for you to join in celebrating the combination of old and new.

You are just hangin' around, no place to go...just chillin'.

You may stroll to another part of RiverCity via the following directions:
  southwest to Downtown west to Black Angel Cemetery
  south to RiverCity Centre

As you look around here, the following places catch your eye:
  slum to City Slum BlueDoor to Hall
  GreenDoor to RoaDs poolhall to Dragonfly's
  NASA to NASA Chateau to Main Hall [Chateau River]
  low brick arch to alley puzzles to Little Shop of Puzzles

NASA (#2109)
A gutted old storefront turned into a chilled out spot for free-thinkers, ravers, and other members of the urban underground. Laser lights, smoke, and powerful amps add to the ambience. The groovy wooden dance floor begs to be abused as does the skate ramp at the back of the shop. There is a Smart bar tucked away under a low rafter beam and pillows spread out in recesses for relaxing. The deep bass, hectic beats, and spacy notes filling the ears of those inside with techno, rattle the old, covered windows at the front of the shop. Welcome to all those ready to open their minds.....ENJOY!
You are swaying to the beat.
You see a sign pointing north here.
Obvious exit: EXIT to Back Alley

Hall (#1933)
Your eyes take a moment to get used to the darkness of the room. The air is fresh and clean here, and faintly scented. The first thing you see is a candle lit on a bookstand in the northwestern corner. Eventually, you discover a stick of incense set in a wooden burner, on an old oak chest. Finally, you see the chair stored under the stairs.
You are here.
Obvious exits:
South to Living room North to Back Alley

Diversity University

(moo.du.org:8888)

Room descriptions: a few details are given. Most of all the descriptions are about what can be done in the MOO. Instructions on how to use the room or the objects in it contained are sometimes given.

Navigation: through names of exits and cardinal points. DU has got also a Web based interface, with a graphical map of its campus.

SAMPLES

```
Architecture building (#666)
....the entrance hall to the Architecture Building.
Besides the other exits out of this room, there is an elevator here.  A
closer look would show the available buttons.

Student Union Lounge (#11)
This is a busy place, frequented mostly by students who're looking for
colleagues to talk to and ways to kill time.  There is an old red couch
in the corner, usually occupied by sleeping students. Several hallways
branch off this center, and large glass doors on the southern wall lead
to the foyer.

Grand Hotel Main Entry (#471)
THIS IS A HUGE LOBBY reminiscent of the majestic hotels from long ago.
There is a split staircase that curves up from the back of the lobby.
At the end of the desk a wrought iron elevator was installed to provide
service for those staying at the hotel.  The hotel contains a
restaurant and conference center as well as a swimming pool and gift
shop. Lining one wall is an elegant *sofa with several *chair(s) facing
it in a semi-circle. In the center of the room you see a large poster
on an easel saying:

     <====== West to the DU Conference Center

MOOteach Learning Center (#1578)
THE MOOteach LEARNING CENTER is dedicated to teaching DU members MOO
basics. It is also for the training and support of teachers who wish to
conduct classes here in MOOspace. Resources are available for the self-
directed study of basic MOO commands, MOO programming and special
teaching strategies and techniques useful for MOOteaching. Enter
DISPLAYS to see some of the resources already available then SHOW #
(number of display item).  Or, LOOK or READ (#) ON (DOCUMENTNAME) the
documents in the room. We hope to be adding new things all the time, so
if you have something to contribute please contact Zak or Jeanne.
To get *Elsie the cow to respond to you, type:  ACTIVATE ELSIE

DU Conference Center Foyer (#15807)
```

A spacious foyer kept in dark shades of pastel colors and white. On the carpetfloor you see shoeprints (the kind you get on freshly vacuumed carpet) that accumulate to a straight track leading from the northern conference room to the one in the south and back. An enormous couch which curves around a round table in a semicircular shape occupies most of the southwestern area.

Conference Room (north) (#15438)
A large, oval shaped room. From the ceiling many fair sized spotlights illuminate the entire room and dip it in warm, bright light. There's a podium at one of the narrower ends of the room. Long rows of chairs, all facing the podium, occupy the rest of the floorspace.

BayMOO

(baymoo.org:8888)

Room descriptions: the rooms are mostly open air areas. Feelings of how the climate, the wind and smells feel like, as well as descriptions of details are given. Some instructions on how to use the objects are also given.

Navigation: through cardinal points as well as name of exits.

SAMPLES

```
*** Welcome to BayMOO! ***
ANSI Version 2.1 is currently active.  Type "?ansi-intro" for more
information.
**************************************************************
There are new news items for you to read.  Please type "news" to get a
summary.
**************************************************************

You have connected as a Guest to BayMOO. We want our guests to feel
welcome here. As a guest, you may create up to 2 objects. You are about
to be asked to give yourself a name and description... This way, you
won't just be some anonymous guest, but yourself. Have fun!  --The
BayMOO Management
[Please type the name you wish to be known as.]
```

(Follows a questionnaire)

```
Guest Login Antechamber (#81)
A small circular decompression chamber for new guests to  catch their
breath and orient themselves to BayMOO space. When you are finished in
here, type south to enter the Aquatic Dome.

Obvious exits: SOUTH to THE AQUATIC DOME and JUMP to Introduction to
BayMOO
You see Population Sign here.
sadisos [Guest], oit [Guest], jupiter [Guest], jenny [Guest], amelina
[Guest], and tagoo [Guest] are here.

The air wavers before you, and Pacifica, a dreamlike figure of a woman
draped in seaweed, seems to hover silently before your eyes.

Welcome to BayMOO, a world for exploring the virtual Bay Area,
Netspace, and Other Worlds.

As a guest, we encourage you to explore the spaces here. You are also
granted a privilege quota of two. This means that you may create two
objects while you are logged in as a guest to BayMOO.
```

When you depart, your created objects must be recycled; but you can get a feeling what it is like to make and use your creations here at BayMOO. We hope that you decide to return and help us build our world!

At any time, type @helpers for a list of people who are willing to help with anything. To contact any of them, type 'page <player> <message>'.

Pacifica beckons you to go SOUTH.

BayMOO Memorial Gardens (#14984)
The gentle breeze brushes against your face as you enjoy the aroma of the fresh cut grass. All around you see memorials of BayMOOers that have passed from this life. Most of all, you notice the silence...
Obvious exits: east to Memorial to Jade, BAY to The Bay Area, and MEADOW to A Mountain Meadow

The Hotel California (#2158)
You find yourself on the outside of the Hotel California. The large court-yard is covered with nice, healthy, green grass. A fountain quietly sprays a light mist into the air. The season's flowers adorn the bottom of the walls where the walls meet the ground. The tightly fitted field-stone walls are covered with Ivy and the motor is covered with that nasty looking green moss. The walls are weather-beaten, but sturdy.
You see Bulletin, Plaque, Scrabble Board, Motorhome, and Grafitti Wall here.
Obvious exits: south to The Bay Area, north to Lobby, northwest to The Cantinia Restaurant, and southeast to Dock

BayMOO Park (#7318)
In the distance, you see a clock tower. (Type look clock).
A lush green park that has a babbling brook flowing thru one area. Lying across the brook are some recent fallen trees that invite you to sit on them. As you walk farther down the path, you come to a large flat stone. One of them invites you to sit on it, as you ponder your thoughts. Then you enter a clump of trees, and see a bench that stretches between two of them. You may sit (type sit stone, log, or bench) and talk quietly to others at your seat. People who remain standing will be heard by everyone in the park.
Look picnic table to see the good stuff to eat. You can always add some of your own food to the supplies.
You see log, stone, bench, and picnic table.
Obvious exits: DOME to THE AQUATIC DOME

A Mountain Meadow (#10423)
You are in a meadow high in the mountains. The sky is pure blue. The air is clean and crisp. The meadow is ringed by snowcapped mountains except for the break through which you arrived. The meadow is strewn with spring wildflowers. At the center of the meadow is a place where the grass has been worn down. Here you see several items. You may <gaze GW> and <look journal>. Also <look EP>. You may <light candle>.

Please <remember> Amanda.

Please <read here> for the order of the ceremony. [held 4/17/92]

You see Granite Wedge, A Journal for Mandy's Friends, Epiphany, Candle for Amanda, White Rose, Another Small Stone, a single rose petal, Small Stone, A small stone, and A Rose from Freedom here.
Obvious exits: MANDY to Blackwood

Ideas and Issues (#99)
A place to discuss ideas and issues about the San Francisco Bay Area

To place a notice in here, just @create #9 named WHATEVER, drop it, then wr*ite you message on the note. Or contact Yea, and he'll add your text to any of the already-existing notes here.

Obvious exits: west to The Bay Area

BlobMOO

(ghoti.stanford.edu:7777)

Room descriptions: Descriptions are about the details and the objects in the room, and general feelings of the place.

Navigation: through cardinal points. An ‘area’ command gives a map of where the player is; a ‘map’ command gives a general map of the environment.

SAMPLES

```
The hole in the ground (#171)
Entering through the rather large hole in the ground, you find yourself
surprized at just how attractive a hole in the ground can be made.  The
hole widens into a large room, with sharp edges and chiselled corners.
The room's walls are bare earth, but peculiarly, nothing looks dirty.
You see only a few tools resting in one corner of the room, plus some
various musical instruments from all over the land are strewn about.
The floor is lined with straw, and there looks to be an enormous pile
of feathers and straw in one corner covered with a sheet -- looks like
a giant-sized bed.
You can go [up] to get out.

The Druidic Forest  (#129)
 The trees are enormous, their dark green branches reaching into the
upper air and blocking out the sunlight almost entirely.  In the dark
shadows of these giant trees, one can imagine any number of strange,
hidden creatures just out of view.  In fact, the occasional streak of
color seen out of the corner of an eye may cause one to whirl about
suddenly and then laugh nervously when nothing unusual appears.  Even
the trees and plants themselves are ominous, seeming to possess an
eerie inner life.  The most striking thing in the forest is the
strange, ash-white tree to the south which seems to dwarf everything
around it and bars all passage in that direction.
To the [west] is the forest trail.  To the [south] is the Millenial
Tree.
You see a shimmer in the air here.
A sprite guest, a pixie guest, a leprechaun guest, and a brownie guest
are asleep.

The Millenial Tree (#390)
The [floor] of the room is strangely carpeted in soft, green grass,
apparently fed by the light from the room's single, round window.  On
it you see nothing.  Near the window, a leaf [hammock] is suspended
between two sturdy branches.  On it you see nothing.  A heavy [wooden
table] occupies the center of the room.  On it you see nothing.  Near
the table is a quaint-looking wooden [rocking chair].  The worn look of
the chair's seat attests to its comfort.  A small [tree] here looks
oddly out of place.
The view from the window is of the tops of the many tall trees that
grow in the Druidic Forest far below.
You see ant class here.
Alyssa is asleep.
```

Corwin's hangout (#649)
This room was built underneath Trakand Plaza. In the ceiling is an opening, which appears to be the bottom of a well.
A bed of hay is in the corner. No one is lying on it. A soft [cushion] is placed directly underneath the opening in the ceiling.
To the [west] is the entrance to a small tunnel. From the tunnel to the [south] comes a foul odor. A [ladder] on the side of the well looks sturdy enough to climb.
You see the Object #900 and Corwin here.

ENTITITES

rocking chair (#570) is owned by Giggles (#421).
Aliases: rocking chair and chair
A generic piece of furniture, *with* sit/stand verbs.

floor (#227) is owned by Giggles (#421).
Aliases: floor
The floor of the room is strangely carpeted in soft green grass.
Alyssa is sitting on it.

LambdaMOO

(lambda.moo.mud.org:8888)

Room descriptions: are about the details and the objects contained. Directions are given in the descriptions of the rooms themselves, almost always using cardinal points. The level of detail is quite fine. The names of the rooms are often aligned with their function and the objects contained.

Navigation: Generally through cardinal points (up and down included). The MOO is so vast that players use their own symbols in their own areas. There is a general map of the MOO, to be found in the living room (#17).

SAMPLES

```
The Living Room (#17)
It is very bright, open, and airy here, with large plate-glass windows
looking southward over the pool to the gardens beyond.  On the north
wall, there is a rough stonework fireplace.  The east and west walls
are almost completely covered with large, well-stocked bookcases.  An
exit in the northwest corner leads to the kitchen and, in a more
northerly direction, to the entrance hall.  The door into the coat
closet is at the north end of the east wall, and at the south end is a
sliding glass door leading out onto a wooden deck.  There are two sets
of couches, one clustered around the fireplace and one with a view out
the windows.
You see README for New MOOers, Welcome Poster, a fireplace, Cockatoo,
lag meter, a map of LambdaHouse, Helpful Person Finder, and The
Birthday Machine here.

The Kitchen (#24)
The kitchen is of a modern design, very large and well-lit, yet still
homey and comfortable.  The walls are covered in beautiful natural-wood
cabinets and the stove is set into a large 'island' counter in the
center of the room.  Over the sink, along the south wall, there are
windows looking out onto the pool and gardens.  At the west end of the
room, there is a little breakfast nook with a table and four chairs;
beyond it to the west is the family room.  A brass ring is recessed in
the tiled floor of the southeast corner, and appears to be part of a
trap door.  There are doors in the north wall leading into the dining
room, a sliding glass door to the south, and a doorway in the northeast
corner leading out into the entrance hall.
You see cookbook, the kitchen sink, Scraps of Paper, vent, a
refrigerator, dishwasher, Microwave, cuisinart, a piece of Saran
Wrap(tm), carrot, and plate of cookies here.

The Dining Room (#28)
This room is dominated by a large pearwood table and six matching
chairs.  On the north wall is a large bookcase, flanked on each side by
French doors leading northwest to the drawing room and northeast to the
smoking room.  The kitchen is visible through a similar pair of doors
to the south, and a large, open archway leads east into the entrance
hall.
```

You see Mastermind Board, Mastermind Instructions, Deck of Playing Cards, Automatic Poker Pot, zoologist, Acquire, Set Game, Quarto, Wooden Chest of Games, Cribbage, Number puzzle, Frand's reversi board, Frand's backgammon board, Snap's connect-4 board, gess board, UpWords board, PenteSet, Ghost game, 'nopoly bank, go board, Game of Hearts, blackbox, Rog's solver for Frand's mind bender, Twister (tm), Solitaire, Scrabble Board, Scavenger Hunt List, Clue, Frand's mind bender, Rubik's Cube, Frand's chessboard, an old coin, and Crazy Eight Ball here.

The Entrance Hall (#19)

This small foyer is the hub of the currently-occupied portion of the house. To the north are the double doors forming the main entrance to the house. There is a mirror at about head height on the east wall, just to the right of a corridor leading off into the bedroom area. The south wall is all rough stonework, the back of the living room fireplace; at the west end of the wall is the opening leading south into the living room and southwest into the kitchen. Finally, to the west is an open archway leading into the dining room.

You see mirror at about head height and Edgar the Footman here.

The Library (#1670)

The library is built in a style reminiscent of old English libraries: very stately, walls completely covered in neatly-arranged books, with a few very comfortable-looking chairs. Each chair is equipped with a footstool, a small side-table and a reading lamp. The carpet is very plush and padded. All in all, the room has an old world kind of charm. Between the stacks to the northwest you see the law section of the library, and there is an alcove just to the west; a massive wooden door in the south wall leads back to the corridor.

You see:

Dusty Bookshelf
Reference Shelf
Geography Shelf
Strange Painting
Literature Shelf
Miscellaneous Shelf
New Submissions Shelf
Helpful Person Finder
Generic Bulletin Board
Ownership Transfer Station
History and Political Science Shelf

Law Section (#7030)

This section of the library seems to be dedicated to MOO law. Most of the shelves contain rows upon rows of thick volumes of case laws, precedents, and interpretations. One set of shelves has been set aside for ballots. Upon those shelves lie several rolled up scrolls of parchment. A few tables and chairs are neatly arranged around the room; one has books scattered upon it. The rest of the library can be seen through the stacks to the southeast. A small sign on a table reads "For information, type `help here'"

You see passed shelf and failed shelf here.

Corridor (#61)

The corridor from the west continues to the east here, but the way is blocked by a purple-velvet rope stretched across the hall. There is a doorway leading into the library to the north, and another doorway to the south.

You see a sign hanging from the middle of the rope here.

Guest Bathroom (#835)

This facility is under construction (quite clearly) by several different contractors who don't know about each other. Someone has started to cover the north wall with carbon black tile, but someone else has started in with clashing sunset brown paneling on the west wall!

The drywall has not even been hung yet on the east wall, and it is still a bare framework of two-by-fours and electric cables. One thick black cable comes in from the wall and disappears into a rough hole in the center of the floor.

There are two bathtubs; one is a verdigris green fiberglass tub with bubble jets, and the second is a cast iron clawfoot tub that is propped up on its end because it does not quite fit in the room. The floor has mostly amethystine painted hardwood towards the north, but someone else has started putting down casino pink slate near the door that exits to the south. One of the workers has left a ladder up against the west wall, and it leads up into a hole in the ceiling.

You see a black cable here.

The Pool (#1428)

The swimming pool is olympic sized, allowing ample room for everyone. This is surprising, considering that the pool deck is a mere 50 feet long... The water is strangely murky; you would expect better care to be taken of the pool. You might find something if you dove down. The sunlight glinting off the water makes you squint.

You see poolsweep, lily pad, thermometer, a bloated corpse, beach ball, and fishing boat here.

You port to The Pool.

Splash!

The Drawing Room (#56651)

A light and airy room with a high ceiling and an ornate crystal chandelier. The walls are decorated with a delicate floral print wallpaper, and a Persian rug with a deep blue ground covers the floor. There is a couch along the west wall, a wing-backed chair and an overstuffed chair across from that, and a window seat. Though it is dark beyond the chiffon curtains covering the bay window, light from the chandelier casts a warm and friendly glow over the entire room. A large stone fireplace dominates the far end of the east wall. A curio table (with a stamp album, Cherry Puzzle Box, and a small bell on it) is near the window. A tremendous aquarium hums and bubbles quietly on the west side of the room. A maple door to the east leads to the Smoking Room; you can see the Dining Room through French doors to the south.

The Smoking Room (#51556)

A dark and spacious room which retains a kind of warmth, despite its crepuscular aspect. The ceiling is high, and a brass gasolier lights the room with an orange, flickering light. Heavy paneling, made of burled maple, lines the walls, and a burgundy, Biltmore carpet covers the floor. The room's seating includes a leather sofa, a bergere armchair, and a pearwood chair. An Edwardian-style fireplace is set into the wall. Set into the wall above the fireplace is a pair of hooks with two fencing sabres resting on them. The single window is framed by full-length draperies. Next to the window is a painting. Off to one side is an elm burl humidor. A long velvet cord hangs down from the ceiling. In the far corner is a pipe rack. Standing against one wall is an antique, cherry bookcase filled with classic works of literature. A silken tapestry hangs from the wall, woven into a beautiful rendition of a medieval castle. It shimmers in the light, attached to two recessed wrought iron hooks. The door to the Drawing Room lies to the west.

A set of French doors to the south lead to the dining room.

You see An albino wolf and groovie heart here.

Editorial Boardroom (#5747)

This is where it all comes together, a small conference room with a low ceiling, paneled walls and the usual flourescent lighting. Here's where all of the big stories get decided. You see a fair amount of ebris remaining from the most recent editorial staff meeting: coffee cups, clippings, and a fair number of typewritten sheets with incoherent scribbling scattered all over the large oak table in the center of the oom.

The Verb Editor and the Note Editor have offices adjoining the conference room to the north and west respectively. The Generic Editor's office is to the northwest, but he's in a really bad mood today so I wouldn't advise going in --- it's not an amazingly friendly place anyway.

The elevator is to the south, the stairwell door is in the southwest corner, the door to the east corridor is in the southeast corner. A State-mandated 2nd-Means-of-Egress door is to the east.

You see ur-Rog's Display Case, Whiteboard, and Lou Grant here.

North Wing Fireside Hall (#47302)

The air here is filled with the smell of spruce and pine, wafting through the air from the fireplace which sits in the center of the room. The room is decorated in a comfortable dark brown, with a varnished hardwood floor and thick dark oak beams supporting the ceiling, and giving the room a general sense of solidity.

Lined up against the western wall stand four or five armored statues, with swords and axes in their metallic hands, looking like obedient servants of Haakon. The fireplace sits in a depression in the center of the room, radiating cozy warmth throughout the room. A wide, circular couch surrounds it, covered in a lush upholstery.

Three large archways lead out to the north, south, and east, with ornate decorations covering the varnished wood. In the southeast corner of the room, a large spiral staircase leads up.

Foyer (#29992)

A lavishly decorated room with thick red carpet. There are tapestries hanging on the walls. To the north is a huge door with ushers standing on both sides of it. There is a podium to the right of the ushers. Sitting on the podium is a registry and a pen. There is an archway that leads back to the south. There are wooden doors to the west and east with signs that say 'GENTLEMEN' and 'LADIES', respectively. There is a marque sign above the door that says: Open House. There is a gold plaque hanging on the wall.

ENTITIES

a black cable (#80563) is owned by grOovY (#47498).
Aliases: a black cable, black cable, and cable
A thick black electric cable. It comes out of a steel punchout in the eastern wall, forms a large tangle on the floor and disappears into a rough-cut hole in the center of the floor.

door (#52810) is owned by Groundskeeper (#80763).
Aliases: door
A narrow wooden door, such as might lead into or out of a closet.

a fireplace (#43212) is owned by GrOoVY (#47498).
Aliases: a fireplace, fireplace, fire, and mantel
An old fireplace taken from an Irish castle that was destroyed in the 16th century. Kindling and logs are piled high along side it. Its mantel is nearly ten feet off of the ground and is supported by two stone gargoyles. On the mantel are several familiar portraits and the unusual sort of bricabrac that you have come to expect, including a cuckoo clock, Graffiti, Big cookie for Graden, Goddess, A cup of tea, a compass, BIg Drink #8, an atlas, and the key to the Pearly Gates. A blazing bonfire rages in the fireplace. In the flames you see Generic Cockatoo Gagging Device, about to burn but somehow still resisting the fire.

dishwasher (#12553) is owned by Blackbriar (#30119).
Aliases: dishwasher
A GE-3950 Dishwasher, Standard ISO 3950-12 Edition. Fully ODA, EISA, and VGA compatible. Cleans All Items That Can Be Cleaned. Removes Food, Unless Too Badly Splattered.

a refrigerator (#1296) is owned by Lambda (#50).
Aliases: refrigerator and fridge
This is a fine, large fridge that looks like it could hold enough to support a fairly large household. There are several little magnets on the door; many of them are being used to hold up notes. Type 'read 6 on fridge' for help on reading what's here.
Contents:
  Vast railroad empire... (#32366)
  I'm kinda busy but I'll clean your stuff... (#10395)
  Adventuring (#277)
  "Nothing We Can Do" (#15251)
  Grand Opening of North Wing and Ballroom (#37266)
  How to use notes on the fridge (#44)
  Group Therapy (#27174)
  See the sites of Lambda! (#34711)
  'If you value your sanity...' (#46478)

```
  MOOers at Work (#75240)
  Character Create Status (#7403)
   ***LAMBDAMOO T-SHIRT OFFER*** (#75014)
  SET! A new game! (#28376) (#24214)
  New Neighborhood! (#31986)
  Roste's Sports Bar (#44880)
  Announcing waffle's book: Your Internet Consultant (#8996)
  new and improved hedge maze (#11030)
  Amazing I Ching garden now open! (#46794)
  [Foobies (#23371):  Be in a Book By Foobies---Read On for Details]
(#20302)
  GrrlTalk (#40264)
  FooBook Mailing List (#12762)
Deutsch sprechen hier (#88084)
  World Wide Web (#7855)
  A Starmap (#62627)
  New Mailing List (#93578)
  My_book (#11150)
  Puzzle room (Lover's Bower) now open (#3810)
  Looking for a set of Trumps (the kind from Amber) (#15407)
  Harmful radioactive mutations for you and your friends! (#15697)
  Mileage (#55420)
  My Mandolin is Looking for a Home (#59273)
  Rumors of space alien visits (#63257)
  New game in the Dining Room (#13076)
  "Foobies is Mirsky (#8212)
  Computer game artist needed for game/demo group. (#89922)
  => OS/2 Warp Peer Support available on *Warp-Mail (#25135) <= (#7993)
  Java-Enabled Moo! (#92594)
  paid announcement (#9738)
  Re: paid announcement (#46595)
  read *Petition:Citizen (#74484)
  Baby Announcement (#90222)
  native american mythology w00F! (#81836)
  Features for Guests! (#11056)
  A public outcry is cast.  Vote no on *b:shutdown, or we will take the
lightbulb out of the fridge!!!, oh and we need milk, bread, and beer..
... (#27741)
  Krups Maximo semi-commercial espresso machine for sale (#13735)
  Read me-World MOO (#66699)
  Rebirth of *IRISH (#65110)
  First Puzzle on MOO! (#134)
[Curator (#88296):  LAMBDAMOO MUSEUM PRESS RELEASE 4/16/97] (#98197)
  Announcement (#110692)
  InfoCorp - A MOO intelligence agency. (#95694)
  Mid-South MOOers (#53975)
  Political Announcement (#69487)
  MOOse - the Forum 2 online community (#69175)
  Lambda Information Center Opens (#6378)
  *comics (#8511)
```

AthenaMOO

(athena.edu:8888)

Room descriptions: the descriptions are pretty poor, with a low level of details. Some rooms totally lack description. This MOO is Pueblo enhances, which means that with an appropriate client (Pueblo) pictures can be visualised.

Navigation: through abbreviations of room names.

SAMPLES

```
The Propylaia (#4702)
Six towering columns support the roof of the outer porch and six more
line the way to the massive gate. Looking up one sees cofferings
between marble beams all painted blue with golden stars. Beyond the
gate one sees the marvels of the Acropolis framed by the supporting
columns of the inner porch.
[NIKE] -Athena Nike
[A] -Acropolis
[AD] -Administration
[W] -Welcome!
[C] -Conference Hall

Social Science Research Center (#1345)
You enter to see a very large, bright room with many video screens as
walls. On the video screens are people from all over the world going
about their lives. You must walk down slightly, so watch your step.
Seats are cushioned and comfortable and are in concentric circles
around the dark green carpeted central area. This is a place where
people learn sociology by doing.
```

The Sprawl

(chiba.picosof.com:8888)

Room descriptions: Descriptions are about functions, brief descriptions and how the places look and feel like. The level of detail is quite fine.

Navigation: mostly through names of exits.

SAMPLES

```
Chiba City Limits (#11)
The lights of Chiba City twinkle in the south east distance.  Street
lights intermittently glow, off to the north.  The smell of inexpensive
liquor and the whine of cathode ray tubes can be heard coming out of
Webster's Bar and URL.  Just over a big tree-covered hill to the south
lies the Village Square of Connections, a nearly-complete experimental
educational environment.  Don't forget to check out the Visitor's
Center.
Obvious exits: north (to 100 Chiba Blvd), down (to Webster's Bar and
URL), west (to 100 Pico Ave), Center (to Visitor's Center), and venue
(to The SenseMedia Venue).

Visitor's Center (#3377)
Welcome to The Sprawl Visitor's Center.  The Sprawl is the World's
First Public Access Web Server and Multi Media MOO.  Publish your own
WWW home page, put graphics and audio in your objects and rooms.  Type
help @request   when you would like a character.
You see Drinking Fountain, Feature Object Closet, The Sprawl Knife,
LaserTag Video Game, Christmas Tree, Picnic Table, and Nordic_Wolf
Punching Bag here.
Orange_Guest (asleep), Dogwood_Guest (asleep), Banana_Guest (asleep),
Spruce_Guest (asleep), Rubber_Guest (asleep), Loblolly_Guest (asleep),
Peach_Guest (asleep), Pear_guest (asleep), Cypress_Guest (asleep),
Eucalyptus_Guest (asleep), Petrified_Guest (asleep), Sycamore_Guest
(asleep), Toon_Guest (asleep), Redwood_Guest (asleep), Pine_Guest
(asleep), Walnut_Guest (asleep), and Elm_Guest (asleep) are here.
Obvious exits: out (to Chiba City Limits), Deck (to The Deck), Restroom
(to Restroom), Survey (to SenseMedia Survey), Kiosk (to Help Kiosk),
and Library (to The Library).

The Library (#4561)
The hardwood floor has a large persian rug in the middle.  Overstuffed
red leather, clawfooted chairs are placed about the room.  Book filled
shelves cover the walls.
You see Looking Glass and "Biochemistry" by Gillianne here.
Obvious exits: out (to Visitor's Center) and glass (to Looking Glass).

Looking Glass (#4993)
A lush green lawn spreads out before your eyes.  A tall hedge blocks
your view in all directions.  The manicured yard is peppered with
holes.  You might be able to make it through the hedge if you don’t
mind a few scratches.
```

```
Obvious exits: Gopher (to Gopher Hole), Worm (to Worm Hole), Rabbit (to
Rabbit Hole), and out (to The Library).
```

DaedaulsMOO

(moo.daedalus.com:7777)

Room descriptions: directions, interesting things in the room, general look of the place.

Navigation: occurs mostly by cardinal points, occasionally by names of exits.

SAMPLES

```
Travelers' Inn (#11)
You are inside a small building used by travelers.  Doors lead out in
several directions. If you are new here, please visit the DaedalusMOO
Information Booth to learn about this MOO.
Obvious exits: out to Outside Caravansary, west to DaedalusMOO
Information Booth, south to Alliance for Computers and Writing, up to
MWSC Virtual Parlor, southwest to CWC95 Center, down to The Coconut
Cafe, east to Virtual Teacher's Lounge, and TCC to TCC Headquarters
You see a newspaper here.

MWSC Virtual Parlor (#2162)
A portable convention center. Lightweight, airy, waterproof. Good to -
50 degrees as long as the surface remains intact. Please do not produce
open flames inside. All warantees are null and void in the event that
the center is exposed to flamable materials or Newtian politics.
Obvious exits: down to Travelers' Inn and north to Convention HQ
You see Recorder2 here.

Convention HQ (#1156)
A routine sort of boardroom with long, plain conference table and
uncomfortable chairs. More water pitchers than people. Since there's
nothing interesting to look at, you're tempted to turn your attention
to issues pertaining to the CCCC annual convention. You find yourself
talking about how things went in DC this year and you toss out ideas
for next year's event. Before you know it you're oblivious to the
terrible chairs and the dull decor.
Obvious exits: south to MWSC Virtual Parlor
You see Recorder3 here.

CWC95 Center (#939)
You find yourself in a computer conference room.  A ring of computers
seems to define the border, but all you'd have to do is scoot a chair
this way or that and you could easily find yourself crossing borders --
and heading who knows where.
Obvious exits: northeast to Travelers' Inn and east to Poster Room
You see Invitation to Netoric's Tuesday Cafe Discussion for 5/23/95 and
Dusty Screen here.

Poster Room (#2838)
These are the posters from the CWC95 Conference MOO session.
Obvious exits: west to CWC95 Center
```

You see MUDtips, KISS, Teaching Composition on a MOO, Sample Class Project, The Writery, VR/WCenter Project, ESL MOO, WebbedMOO Teaching, WebbedMOO Space, Professional Uses of MUDs, Scholarly Uses of MUDs MOOs and IRC, and Tips for Structured Meetings here.

Alliance for Computers and Writing (#1150)
You are in a small ampitheater with a grassy floor. The area is small enough for everyone to hear each other, large enough to contain all interested parties. The grass is beautifully tended so that the grass roots are healthy, giving the leaves a velvety look.
You see Recorder here.
Obvious exits: north to Travelers' Inn

Through a miracle of cyberspace, you slip through glass, air and wind to a spot with a tropical air. The tradewind is fresh, the air moist and the fragrance of wild ginger pushes you into a different pace.
The Coconut Café (#1710)
Where the lights never go out -- and where you can take off your shoes, kick back, and relax with great tropical drinks and good friends.
Obvious exits: out to Travelers' Inn, up to TCC Penthouse, and wired to 10101
Wired Beltway

ENTITIES

Scrawled Letters (#4276) is owned by cin (#2822).
Aliases: Scrawled Letters, scrawl, scrawled, and letters
there seems to be a hint of some sort here...

a stage (#241) is owned by Jeff (#182).
Aliases: a stage and stage
A small stage that is just large enough for you to dance on.

MediaMOO

(purple-crayon.media.mit.edu:8888)

Room descriptions: almost only about the functions. Objects and a few details are sometimes described.

Navigation: through names of exits, a map is available

SAMPLES

```
media.mit.edu (#367)
You are in a maze of little, twisty passages.
You see the back of a computer screen here.
Obvious exits: down to mit.edu, purple-crayon to purple-
crayon.media.mit.edu, mfulu to mfulu.media.mit.edu, out to Systems
Programming, cherupakha to cherupakha.media.mit.edu, term1 to
term1.media.mit.edu, spot to spot.media.mit.edu, Peerssa to
Peerssa.media.mit.edu, and groovus_maximus to
groovus_maximus.media.mit.edu
```

```
Curtis Common (#7813)
                    _____________________________         ____________
                   |                             |       |            |
          .--------|_____________      MEDIA      |-------|  Library   |
         /         | E&L Garden|       LAB       |      NE            |
        /       nw~~|_______   __|_________________|       |____________|
       |                    N                                    |       +
_______|                                                         |_______+_
Media    W                          ____                       E  Media  +
_Ave____                          /  / ~ ~\ \     CURTIS            __Ave___+_
     ___|____       *             |~ ~ ~ |      COMMON       _______|____   +
    |        |     ***            \ \_~__/ /                 | TechnoRet  |  +
    |        |     *|*              ==  ==                ESE  Bar & Grill|   +
    |        |                                              |_____________|
    |Theater |                                                _______|_
    |        |         _ SSW_            ___ SSE ____          |Poetry |
    |         SW      |      |          |   Book   |      SE  Reading|
    |________|--------|_Deli_|----------|_Reviews__|---------|_Room__|
```

```
You see Fountain, BLUEPRINT#1, BLUEPRINT#2, Portrait of Frankie, Chain-
Story, and purple frisbee here.
```

```
The TechnoRhetoricians' Bar and Grill (#2132)
A slovenly, comfortable hovel where the patrons discuss the rhetorical
implications of every little thing, including the shifting dunes of
crumbs and peanut shells that ripple across the floor. No one here is
daunted by triviality.
Obvious exits: north to the panopticon, west to Curtis Common, up to
CWTA Outpost, and cafe to RhetNet Cafe
You see MacXVI, Lou the Bartender, Pony, Isocrates, Derrida, toolbox,
Barthes, and ProfProj here.
Glenn, MC, beckster [New Mamma], sm, and Eric are here.
```

Poetry Reading Room (#6804)
You enter a room built of adobe shaped into thick walls and rounded surfaces. A fire of pinon [sic] and cypress burns and crackles in one corner. Navajo rugs cover the floor and walls. There are benches built into the walls. Pillows and cushions of various sizes, enough for everyone, are scattered on the floor and benches.
On a bookshelf near the fire is a note explaining the room <look note> and some empty poetry journals <look journals> waiting to be used.
You see Hymn to Mercury and a fez here.
Obvious exits: west to Moon and out to Curtis Common

Library Foyer (#4298)
This is the entrance (and center) of the MediaMOO Generic Library. It looks very library-like indeed, with a brightly lit interior and many directions in which to go.
Obvious exits: west to Library (room of Things), north to Library (room of Rooms), east to Library (room of Player Classes), south to Library (room of Features), lab to The E&L Garden, and up to MediaMOO Political HQ.
You see Sign about Library Policy, Library Comments board, and transfer here.

The E&L Garden (#11)
The Epistemology and Learning Group garden is a happy jumble of little and big computers, papers, coffee cups, and stray pieces of LEGO.
Obvious exits: library to Library Foyer, atrium to Third Floor Atrium Landing, and common to Curtis Common
You see a newspaper, a Warhol print, Sun SPARCstation IPC, Projects chalkboard, Research Directory, Constructionist Flag, and Train Transfer here.

Third Floor Atrium Landing (#93)
The hallway here opens up into a four-story atrium. You are on the third floor. Over the railing and across the atrium, you see a mural of grey squares with colored lines between them.
Obvious exits: lego to The E&L Garden, cw to VisMod Hallway, ccw to E&L Hallway, stairs to Third-floor Stairwell, elevator to elevator, and kitchen to Media Lab Kitchen
You see an elevator call button and Map of Media Lab here.

Elevator (#332)
You are inside the Media Lab elevator. The walls are dull steel and the floor is grey rubber with raised circular bumps. You see push buttons labeled LL, G, 2, 3, 4, Roof, and 6. The floor indicator shows 3. The elevator is stopped and the doors are open.
Obvious exits: elevator to elevator

Sixth-floor Atrium (#182)
A mist-filled space. The topology of this floor is unclear.
Obvious exits: library to The Library, down to Media Lab Roof, alum to Alumni Hallway, south to Ballroom Foyer, and elevator to elevator
You see an elevator call button here.

ENTITIES

Map of Media Lab (#10001) is owned by Ninja_Librarian (#1751).

```
Aliases:  Map of Media Lab and map
          MAP OF THE MEDIA LAB
      Type:  read <symbol> on map

          Symbol     Floor
           LL       Lower Level
           G        Ground Floor
           2        Second Floor
           3        Third Floor
           4        Fourth Floor
           R        Roof
           6        Sixth Floor

   If you leave this room, type:   read <symbol> on #10001
```

AICoreMOO

(aisun2.ai.uga.edu 2357)

Room descriptions: functions, mission statements, few details about the objects and the environment

Navigation: by cardinal points

SAMPLES

```
Possible Worlds (#61)
This is the starting point for a virtual community dedicated to the
study and implementation of AI systems.

A with the historical backdrop of Whitehead and Russel in their
'Principia Mathematica', Goedel's Incompleteness Theorum opened the way
for a variety of formal methods:  a theory of recursive functions,
turing machines, lambda calculus and even systems for rewriting
strings.
You see Welcome Sign and Generic Told-Text Driven Bot here.
               AI Fields                down: Virtual R&D Labs
             -----        N        -----
   usfl cmptrs   W  *  E undrstndng intl
             -----        S        -----
                      Library

Library (#133)
A collection of documents, books, and other reference materials
relevant to the studies being conducted at AICore.

        "We can only see a short distance ahead, but we can
         see plenty there that needs to be done."
                                       - Alan Turing

You see Mail Distribution Center, moo Oddities, 'Zine Rack, and Elvis
the Librarian here.
Kat (asleep), cricket (asleep), and edm (asleep) are here.
             Possible Worlds              down: Vic's Study
        Useful computers   N  undrstndng intllg
 Museum MetaRoom W  *  E   Cr Rvw Brd
                         -----        S        -----
                    AICore Ontology Studies

Understanding Intelligence (#97)
  "Artificial intelligence is the computer modeling of human mental
abilities. We see it as an interdisciplinary field where computer
science intersects with philosophy, psychology, linguistics,
engineering, and other fields.   "
You see psyBotA, psyBotC, and psyBotB here.
Giuseppe (asleep) and Minister (asleep) are here.
                      Virtual_Poetics
```

```
            AI Fields      N        -----
 Possible Worlds W  *  E      -----
              Library       S        -----
                Core Review Board
```

Useful computers (#94)

One of the goals of AI research is to improve the pragmatic utility of our computation resources.

Programs using methods of symbolic computation perform useful tasks, which do exhibit intelligence. An open question remains as to whether it is the program's or the programmer's . . .

Systems designed to facilitate storage, transfer, and recall of sets of facts and relations augment the user's intelligence. More sophisticated tools for graphic visualization of complex concepts or defeasible inferencing may allow certain types of intellectual feats previously thought impossible.

You see News Database and Logic Graphs and Matrix Algebra here.

```
        Generic Logic Graph Room
       Hypermedia      N      AI Fields
   Turing Room   W  *  E Possible Worlds
   PrgrmmngLnggs    S
                           Library
```

AI Fields (#188)

This is a newly created area of discussion and study . . . perhaps a discussion of Artifical [sic] Life, or Minimally Intelligent Things is a starting point for this topic.

Everyone is encouraged to '@dig' whatever field of AI that is of interest.

You see Generic 3D World and Common Gateway Interface here.

Worm (asleep) is here.

```
            Cellular Automata
   Neural Network    N        -----
          CASE       W  *  E  Lgc Prgrmmng
 Useful computers   S  undrstndng intllg
            Possible Worlds
```

BioMOO

(http://bioinfo.weizmann.ac.il:8001)

Room descriptions: materials and textures, some architectural descriptions.

Navigation: through a Web interface (BioGate system). Mostly by cardinal points, sometimes by names of exits. A Web and ASCII map is available.

SAMPLES

```
The Lounge (#11)
A large, silent, dimly illuminated room with lots of people snoring
their real lives away...
A door to the south leads out to the Central Room. (type 'south' to
exit the Lounge; omit the quotes when you enter the command).
A big sign here reads: Type 'tutorial' to learn the basics of MOO, or
if you have a web browser available, type 'web' for a web tutorial.

The Central Room (#126)
A very large, circular room, its ceiling a transparent dome through
which sunlight streams in.
A large archway leads west into a foyer.
There are doors leading in all directions, labelled with tasteful
signs.

You see  the Research Directory,  a public bulletin board,  a
Jobs/Postdocs bulletin
board, a Lost and Found Box, a tour dispenser (td), a statue of a
rodent pointing south to the zoo, and
an animatronic rhesus macaque (rhes).

Obvious exits: west (to the Foyer), north (to the Lounge), northeast
(to the Seminar Room), southeast (to the Bio Center Lab Tower Lobby),
northwest (to the BioMOO Central Library), southwest (to the Special
Interest Area (sia)), south (to the Bio Center Office Tower Lobby), and
east (to the GCG Help Desk).

The Bio Center Office Tower Lobby (#3558)
You are in the bottom of a cylindrical tower that stretches up well
beyond your sight. The floor and walls of the tower are paneled with
white marble. Randomly spaced along the walls are balconies leading
into offices (which you can reach using the office directory you see
here).
The floor is composed of circular disks, each two meters in diameter,
that wobble slightly as you step on them.
A door to the north leads into the Central Room. Another to the south
leads out onto the Bio Center Lawn.

You see  an Office Directory and  a sign.

Obvious exits: north (to the Central Room) and south (to the Bio Center
Lawn).
```

The Bio Center Lawn (#312)
An expanse of greenery. You see patches of colourful flower beds and hear the drone of bees.
The turf rolls easily from the Biological Center southwards to a calm lake.
To the west you can see some woods. To the east, you see the entrance to the ZOO grounds, and faraway, a
river flows placidly.

Nobody else is here.

Obvious exits: west (to the woods), south (to the shore of the lake), east (to the Zoo Gate), northwest (to the Campus Garden), and north (to the Bio Center Office Tower Lobby).

The BioMOO Central Library (#633)
A vast octagonal room whose entire north side is a large panelled window looking north. Angled parts of the walls near the ceiling and floor give the impression of facets.
The room is dominated by a spiral staircase in its center, leading down to the generic objects display area, and up to the audiovisual center. Doors are set into each wall, but only three are open.
A door to the south opens to the BioMOO/MOO References room, one to the west enters the Biology Shelves, and one to the southeast leads back to the Central Room.

You see a table covered with sheets of paper, a sign, a map, and a Users directory
(ud).

Nobody else is here.

Obvious exits: southeast (to the Central Room), down (to the Generic Objects Display area), up (to the AudioVisual Center), south (to BioMOO/MOO References), and west (to the Biology Shelves).

ENTITIES

The table covered with sheets of paper (#2044)
A rectangular table covered with pages. The nice lettering on them invites you to read...
1 - Norm: Journal Clubs on the BioMOO Pamphelet
2 - El gato: Note on grammar
3 - david: On Objects(OO)
4 - mouse: Note for new users

PennMOO

(ccat.sas.upenn.edu:7777)

Room descriptions: descriptions of the inside are of a good level of detail. Instructions on how to use the room and its contents are gives.

Navigation: through cardinal point, names of exits, elevator, map. An ASCII map is available, some directories of rooms, and a command “ways” which gives the possible directions.

SAMPLES

```
Virtual Apartment (#2133)
This 24'x24' room needs to be decorated.  The walls are covered with a
wallpaper of pink tulips and green leaves on a soft yellow background.
You are invited to turn this room into a livable and welcoming
apartment. You may add furniture to this room by writing descriptions
of appropriate furniture. Furniture can also be removed if
inappropriate. This is your virtual apartment, so be creative! You may
add or remove furniture by typing these commands and following the
directions:
          addfurn <name of thing>
          rmfurn <name of thing>
You see picture, Air conditioner, Study Desk, chair, sofa, flower,
plant, cd player, cd's, more cd's, bed, refrigerator, computer,
television, books, cd player, poet's chair, souvenir, stereo,
microwave, chaise longue, marble-topped coffee table, and doily.
You see Yellow Lamp and bagel.

Fara's Basement (#1539)
It's a slightly musty basement, with bright fluorescent lights shining
down on a pile of stuff.  All sorts of stuff.  It seems that Fara is
planning to have a yard sale, or maybe she just likes to save things
which might be useful someday.

If you would like to add to this pile of stuff, type-->  addfurn <name
of thing>  <--and follow directions.
You see Big Table, Baby Chair, bike, gloves, PC, Redskins helmet,
loveseat, AMEX bill box, tricycle, china, Recliner, bulkpack, rose,
Post Card, miniature red corvette, a notebook computer, flowers, tent,
old  sleeping bag, and books.
You see old boombox.

Locust Seminar Room (#268)
You are in a large conference room, set up with three clusters of
chairs for small group discussions.  People seated at one of these
clusters can talk to each other without being heard in the rest of the
room.  People standing can talk to the entire room.
You see red chairs, blue couches, and yellow table.

Furness Library (#295)
```

An elaborate hundred-year-old building of red sandstone, in the Queen Anne eclectic style. Named for its architect Frank Furness, this is a place both to explore, and to make use of moo and internet library resources.

The library is still under construction, but please look around.
You see PennMOO kiosk here.

APPENDIX D. Virtual Campus examples

Designer Class

Several classes of players are available in the Virtual Campus. Each class allows characters to access special commands, only available to that class and offspring. To explain the implementation of the designer class of player, I need to introduce a few technical details about how the player hierarchy is actually defined.

In the Virtual Campus, the hierarchy of registered characters, or *players*, is (as displayed by the command *@classes generics*):

```
generic player (#6)
    Generic Mail Receiving Player (#91)
          Frand’s player class (#90)
                generic builder (#4)
                      generic programmer (#57)
                            generic designer (#575)
                                  generic administrator (#56)
```

A generic administrator can access all the commands attributed to the previous classes (designer, programmer, builder, and so on).

A *generic builder* (#4), and its offspring, can build new entities, starting from existing ones, with the command:

```
@create <parent> named <new object>
```

where `<parent>` stands for the original entity that is being cloned (eg. a note, #9) and `<new object>` is the name of the referent (eg. banner).

Programmers can use all the commands available to builders, plus they can change, add and remove verbs and properties of virtual entities. They can therefore change, add and remove activities and reactions, and change the referent of an object by modifying its description, help text, and other related information.

The *designer class* has been introduced to assign a set of commands, or Design Speech Acts, to users who wanted to experiment design in the MOO. The designer class assigns

special features to users belonging to it, and it represents a simple way to gather under one class all the users who are experimenting design in the MOO.

Users belonging to the class of designers, can:

- use all the commands available to the parent classes (for example, to builders and programmers);
- create new objects using Design Speech Acts;
- access a special entity *design studio*;
- use DSAs to modify activities and reactions;
- participate to a MOO discussion forum regarding design issues (the list *designers);
- interrogate the MOO database regarding entity (A, R, Ref);
- add special features to their own entities.

Designers can access verbs and properties directly (eg. with the `@program` or `@prop` commands), using Design Speech Acts, and within the *Design Studio*, an editor that facilitates design operations.

This class is an attempt to define the designer role within the Virtual Campus, by defining a set of DSAs, special entities, and procedures (eg. entering the Design Studio to design), in accordance with the architecture of the environment and the software structure. The (A, R, Ref) model provides information for the implementation of these DSAs, entities, and procedures.

Design Speech Acts

Following the DSAs already implemented (and completed) in the Virtual Campus.

#4:@sketch

```
#4:@sketch   any none any
 1:  "@sketch <prototype> <newname>";
 2:  "This command selects from a prototype of a room, and asks a
series of questions.";
 3:  "A set of prototypes must be used for the verb to work. It creates
a new room, sketched, with parent ($prototype). Find the prototypes in
$builder.prototypes.";
 4:  set_task_perms(player);
 5:  nargs = length(args);
 6:  if (!args)
 7:    player:tell("Usage: @sketch <prototype> <name>");
 8:    player:tell("To see a list of prototypes, type: @prototypes");
 9:    return;
10:  elseif (!args[2..nargs])
11:    player:tell("You must enter a name for the new room.");
12:    while ((newroom = $command_utils:read()) == "")
```

```
13:      player:tell("Please, enter a name for your new room.");
14:    endwhile
15:  else
16:    newroom = $string_utils:from_list(args[2..nargs], " ");
17:  endif
18:  prototype = args[1];
19:  if (!(prototype in this.prototypes))
20:    player:tell("You have to name a specific prototype to sketch
from.");
21:    player:tell("Type @prototypes to see what's available.");
22:    return;
23:  endif
24:  if (instr = $command_utils:yes_or_no("Do you want to read the
instruction on @sketch?"))
25:    descr = {$help:get_topic("@sketch", $help)};
26:    player:tell_lines(@descr);
27:  endif
28:  player.location:announce(player.name, " begins @sketching ",
newroom, " as ", $string_utils:a_or_an(prototype[2]), " ", prototype, "
area.");
29:  player:tell("You begin @sketching ", newroom, " as ",
$string_utils:a_or_an(prototype[2]), " ", prototype, " area.");
30:  "Here the questions start";
31:  correct = "no";
32:  while (correct[1] != "y")
33:    player:tell("Type in a description for this ", prototype, "
area:");
34:    description = {@$command_utils:read_lines()};
35:    player:tell("What alias would you like to use for this area?
(leave empty if none)");
36:    aka = $command_utils:read();
37:    player:tell("Type in the name of the exit from ",
player.location.name, " to ", newroom);
38:    if (!aka)
39:      aka = newroom;
40:    endif
41:    while ((exit = $command_utils:read()) == "")
42:      player:tell("You must enter a name for the exit.");
43:    endwhile
44:    parent = player:my_match_object(prototype);
45:    if (parent == $failed_match)
46:      player:tell("Sorry, this prototype, ", prototype, ", has not
been defined yet.");
47:      return;
48:    endif
49:    "   End of questions, now confirm.";
50:    player:tell("You have defined the basic information for your ",
newroom, ". Here is the information you have entered:");
51:    player:tell("Name: ", newroom, ", with alias ", aka);
52:    player:tell("Parent/prototype: ", prototype, "(", parent, ")");
53:    player:tell("Exit from ", player.location.name, " to ", newroom,
": ", exit);
54:    player:tell("Description:");
55:    player:tell_lines(description);
56:    player:tell("Is this information correct? (yes or no)");
```

```
57:    correct = $command_utils:read();
58:  endwhile
59:  "    Now create it!";
60:  new = player:_create(parent);
61:  new.name = newroom;
62:  new.aliases = {aka};
63:  "    Partially copied from #4:@dig.";
64:  do_recreate = !player:build_option("bi_create");
65:  to_ok = $building_utils:make_exit(exit, player.location, new,
do_recreate, $exit);
66:  from_ok = $building_utils:make_exit("out", new, player.location,
do_recreate, $exit);
67:  "    Room and exits have been created.";
68:  new.description = {@description};
69:  player:tell("You have now a new room, ", new, ", named ",
new.name, " (aka ", @new.aliases, "), and with parent ", prototype,
".");
70:  suspend(4);
71:  this:teleport(player, new);
72:  if ($improved in $object_utils:ancestors(parent))
73:    player:tell("");
74:    player:tell("The details here are:");
75:    message = player.location:list_details();
76:    player:tell(@message);
77:    details = parent.details;
78:    for det in [1..length(details)]
79:      detail = new:match_detail(details[det][1], 1);
80:      player:tell("");
81:      player:tell("Detail: ", details[det][1]);
82:      if (answer = $command_utils:yes_or_no("Do you want to remove
this detail?"))
83:        new.details = parent:rem_detail(new, detail);
84:      endif
85:    endfor
86:  endif
87:  suspend(4);
88:  player:tell("");
89:  "Does the player want to clone objects from the $prototype?";
90:  contents = parent.contents;
91:  if (contents)
92:    player:tell("The ", prototype, " prototype you have chosen
contains some special objects, that you can clone.");
93:    for a in [1..length(contents)]
94:      player:tell("Object: ", contents[a].name, "(", contents[a],
")");
95:      if (answer = $command_utils:yes_or_no("Do you want to clone
this object?"))
96:        newobj = player:_create(contents[a]);
97:        if (newobj == E_PERM)
98:          player:tell("Sorry, the object doesn't seem to be
fertile.");
99:        else
100:          newobj.name = tostr(contents[a].name, "_clone");
101:          player:tell("You have a clone of ", contents[a], ", with
number ", newobj, ", named ", newobj.name, ".");
```

```
102:          move(newobj, player.location);
103:        endif
104:      endif
105:    endfor
106:    player:tell("You can rename your objects with '@rename <object>
to <new_name>'.");
107:  endif
108:  player:tell("");
109:  player:tell("");
110:  player:tell($string_utils:center(" SKETCH FINISHED ",
player.linelen, "-"));
111:  "Last modified Fri Jun  5 15:31:55 1998 EST by Creeper
(#101@Virtual_Campus).";
```

#184:@copydetail

```
#184:@copyd*etail   any (at/to) any
 1:  "Usage:  @copyd*etail <detail> to <room>";
 2:  "Create a copy of <detail> in a new room, which has among its
parents, at least, the improved room.";
 3:  "";
 4:  "Only the owner or a wizard can copy the details in a room.";
 5:  if (!$perm_utils:controls(player, this))
 6:    return player:tell(this:no_perms_msg() || "You are not allowed
to do that here.");
 7:  endif
 8:  if (!dobjstr || !iobjstr)
 9:    return player:tell("Usage:  ", verb, " <detail> to <room>");
10:  elseif (!(source = this:match_detail(dobjstr)))
11:    return player:tell("There is no detail by that name (", dobjstr,
").");
12:  endif
13:  to_room = toobj(iobjstr);
14:  must_be = #184;
15:  if (!$object_utils:isa(to_room, must_be))
16:    player:tell("The room you are copying the detail to must be a
child of, at least, ", must_be.name, "(", must_be, ").");
17:    return;
18:  endif
19:  names = $building_utils:parse_names(dobjstr);
20:  dnames = $list_utils:remove_duplicates({names[1], @names[2]});
21:  if (detail = to_room:match_detail(dnames[1]))
22:    if (dnames[1] in to_room.details[detail])
23:      return player:tell("There already appears to be a detail
called \"", dnames[1], "\" in ", to_room.name, ".");
24:    endif
25:  endif
26:  if (typeof(to_room.details) != LIST)
27:    player:tell("*** Cannot add detail -- details list is corrupted!
***");
28:    return player:tell("type '@go ", to_room, "' and '@fix here' to
correct this problem.");
29:  endif
30:  copied = this:copydetail(this, to_room, dnames);
31:  if (!copied)
```

```
32:    player:tell("Mmm..... for some reasons I don't seem to be able
to copy that detail.");
33:    return;
34:  endif
35:  player:tell("Detail \"", this.details[source][1], "\" copied as ",
copied, ", to ", to_room.name, ".");
36:  player:tell("Please, check the detail you have copied carefully.
You may need to add some properties and/or verbs needed by the detail,
to perform specific actions.");
37:  "Last modified Fri Jun  5 13:39:33 1998 EST by Creeper
(#101@Virtual_Campus).";
```

#575:@copyobject

```
#575:@copyob*jects   none none none
 1:  "Does the player want to clone objects from the $prototype?";
 2:  parent = $office;
 3:  contents = parent.contents;
 4:  for a in [1..length(contents)]
 5:    player:tell("Object: ", contents[a], contents[a].name);
 6:    player:tell("Do you want to clone this?");
 7:    answer = $command_utils:read();
 8:    if (answer == "yes")
 9:      newobj = player:_create(contents[a]);
10:      newobj.name = tostr(contents[a].name, "_clone");
11:      player:tell("You have a clone of ", contents[a], ", with
number ", newobj, ", named ", newobj.name, ".");
12:    endif
13:  endfor
```

#575:copy_object

```
#575:copy_object   this none this
 1:  "Copies an object to a new one, as a clone.";
 2:  "Returns the number of the cloned object.";
 3:  source = args[1];
 4:  player:tell("source: ", source);
 5:  new = player:_create(source);
 6:  player:tell(new);
 7:  return new;
```

#4:@prototypes

```
#4:@proto*types   none none none
 1:  set_task_perms(player);
 2:  player:tell("Available prototypes for the @sketch command are:");
 3:  player:tell_lines($string_utils:english_list(player.prototypes, 4)
+ ".");
4:  "Last modified Fri Jun  5 15:38:45 1998 EST by Creeper
(#101@Virtual_Campus).";
```

#213:@addfurniture

```
#213:@addfur*niture   any any any
```

```
 1:  if (this:is_authorized(player))
 2:    if (argstr)
 3:      if (this:is_furniture(argstr))
 4:        player:tell("Sorry, there is already something here by that
name.");
 5:        return;
 6:      endif
 7:      player:tell("Please describe ", argstr, ".");
 8:      descrip = $command_utils:read();
 9:      s = $command_utils:yes_or_no("Is this object sittable? ");
10:      if (s)
11:        player:tell("Type in a self-message for when a person sits
down.");
12:        player:tell(" Example: \"You sit down at the %f and make
yourself comfortable.\" (%f will get substituted with ", argstr, ").");
13:        sit = $command_utils:read();
14:        player:tell("Type in a public message for when a person sits
down.");
15:        player:tell(" Example: \"%n sits down at the %f.\" (%n will
get substituted with the sitter and %f with ", argstr, ").");
16:        osit = $command_utils:read();
17:        player:tell("Type in a self-message for when a person stands
up.");
18:        player:tell(" Example: \"You rise from the %f and stretch.\"
(%f will get substituted with ", argstr, ").");
19:        stand = $command_utils:read();
20:        player:tell("Type in a public message for when the player
stands up.");
21:        player:tell(" Example: \"%n rises from the %f.\" (%n will
get substituted with the sitter and %f with ", argstr, ").");
22:        ostand = $command_utils:read();
23:        player:tell("Type in a message for the room's description
when people are sitting on ", argstr, ".");
24:        player:tell(" Example: \"%crowd %is sitting at %f.\" (%crowd
will get substituted with the people sitting there, %is with either
\"is\" or \"are\" and %f with ", argstr, ").");
25:        sitting = $command_utils:read();
26:        s = {sit, osit, stand, ostand, sitting};
27:      else
28:        s = {};
29:      endif
30:      this.furniture = {@this.furniture, {argstr, {}, {}, descrip,
s}};
31:      this:announce_all(player.name, " sets up a ", argstr, " in the
room.");
32:    else
33:      player:tell("To add a new object you must give it a name:
'@addfurniture <name>'");
34:    endif
35:  endif
36:  "Last modified Fri Jun  5 15:44:25 1998 EST by Creeper
(#101@Virtual_Campus).";
```

#213:@rmfurniture

```
#213:@rmfur*niture   any any any
```

```
 1:  if (this:is_authorized(player))
 2:    if (!argstr)
 3:      player:tell("To remove something you must name it:
'@rmfurniture <name>'");
 4:      return;
 5:    endif
 6:    if (found = this:is_furniture(argstr))
 7:      this:announce_all(player.name, " removes ",
this.furniture[found][1], ".");
 8:      this.furniture = listdelete(this.furniture, found);
 9:    else
10:      player:tell("Sorry, there is no furniture here by that
name.");
11:    endif
12:  endif
13:  "Last modified Fri Jun  5 15:46:45 1998 EST by Creeper
(#101@Virtual_Campus).";
```

Area Prototypes

Following the area prototypes designed in the Virtual Campus. Some verbs and properties of these prototypes, have been inherited from parent entities, programmed by other MOO users. Authors of verbs maintain their copyright, as signalled by the last line of the verb. If an author's name is not present, the verb should be considered part of the original LambdaCore Database.

$private

```
@archive #434 with verbs and properties
----- Beginning of @archive #434

@create #206 named Private Room Prototype,private

;#434.("html_desc_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("details") = {{"filing cabinet"}, {"trees"}, {"fountain"},
{"table"}, {"clock"}, {"other chairs", "chairs"}, {"window"}, {"owner's
desk", "desk"}, {"owner's chair", "chair"}}
;#434.("desc_dmsgs") = {0, {"*call*", "tree_growth"}, "A spring of cool
water is coming out from it.", 0, {"*call*", "clockface"}, {"These
chairs are for the guests of this office.", "Please, sit on one."},
{"*call*", "window_look"}, "Just the owner's desk.", 0}
;#434.("take_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("otake_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("seats") = {0, 0, 0, 0, 0, 10, 0, 1, 1}
;#434.("sitting") = {}
;#434.("sitting_seats") = {}
;#434.("capacity") = {20, 0, 0, 2, 0, 0, 0, 0, 0}
;#434.("contained") = {}
;#434.("containing_details") = {}
;#434.("sit_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("osit_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("sit_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("osit_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
```

```
;#434.("stand_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("ostand_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("sitting_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("put_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oput_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("put_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oput_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("get_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oget_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("get_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oget_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("contained_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("help_msg") = "Under construction."
;#434.("touch_dmsgs") = {0, 0, 0, 0, 0, 0, 0, "It's cold.", "It's
marble, pretty cold."}
;#434.("otouch_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("smell_dmsgs") = {0, 0, 0, 0, 0, 0, 0, "It smells like glass
polisher.", 0}
;#434.("osmell_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("listen_dmsgs") = {0, 0, 0, 0, 0, 0, 0, "You can hear some
noises...", 0}
;#434.("olisten_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("taste_dmsgs") = {0, "It's a minty flavour.", "Nice, clear, and
fresh.", "Nice, clear, and fresh.", 0, 0, 0, "YUUUUUMMMM!", 0}
;#434.("otaste_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("opened_containers") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("invisible_contents") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("open_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oopen_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("close_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oclose_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("open_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oopen_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("close_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("oclose_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("opened_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("closed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0}
;#434.("editing_buffer") = ""
;#434.("editing") = {1, "desc_dmsgs"}
;#434.("help") = {#434}
;#434.("entrances") = {#890}
;#434.("blessed_object") = #-1
;#434.("blessed_task") = 440512195
;#434.("exits") = {#447, #889}
;#434.("invited") = {}
```

```
;#434.("instruction") = {"", "$PRIVATE area prototype", "----------
----------------------------", "This set of instructions is the
default for the prototype $private.", "You can edit your instructions
once you have @sketched a new room with this parent, by typing '@edit
<your_room>.instruction'.", "----------------------------------------
-", "A Private Area Prototype has the following details:", "- the
owner's chair; a comfortable chair where the owner of the room can
rest;", "- the owner's desk;", "- other chairs; you can set how many by
typing: @sets chair is <number>", "- a window; this looks over a room
you designate (type: @window is <room>, where <room> is where you want
the window to look). Type 'look window' to see who is in the room on
the other side of the window. The room where the window is looking at,
is stored in the <room>.window_target. You can change that with
'@window is <room_target>'.", "- a fountain;", "- some trees; try 'look
trees';", "- a table;", "- a filing cabinet;", "- a clock; to read the
time type 'look clock';", "", "NOTE: To copy a detail from one room to
another, try @copydetail (see help details and @copydetail for more
information)."}
;#434.("prototype") = #434
;#434.("key") = 0
;#434.("aliases") = {"Private Room Prototype", "private"}
;#434.("description") = {"This room is to be used as a prototype for
private areas, such as offices, studios and workshops, other personal
rooms.", "Type: @instr here, to see how to use this room.", "You see
the owner's chair, and the owner's desk against the wall, on it a
clock, a window which looks outside, somewhere in another MOO room, and
some other chairs around a table, a filing cabinet where to put your
documents.", "In a corner, in a place which seems another dimension,
you see a fountain under some trees."}
;#434.("object_size") = {11614, 896774413}
;#434.("web_calls") = 99
;#434.("last_modified") = 895819643
;#434.("creation_date") = 874047675
@prop #434."window_look" {"From the window you see "} "rc"
@prop #434."window_target" #61 "rc"
@prop #434."tree_height" 6 "rc"

@verb #434:"window_look" any any any "rxd"
@prog #434:window_look
"Looks into the target window (to_room), and gets a list of contents.";
source = player.location;
to_room = source.window_target;
cnames = $string_utils:name_and_number_list(to_room.contents);
message = tostr(@source.window_look, "[watching ", to_room.name, ", ",
to_room, "]: ", cnames, ".");
return message;
.

@verb #434:"@window" none is any "rx"
@prog #434:@window
"Sets the room where the window is looking at.";
target = toobj(player.location.window_target);
room = target.name;
newtarget = toobj(args[2]);
if (!args[2])
  player:tell("Usage: @window is <newtarget>");
```

```
  player:tell("Your window actually watches ", room, " (", target,
").");
  return;
endif
if (!valid(newtarget) || !($room in
$object_utils:ancestors(newtarget)))
  player:notify(tostr(newtarget, " doesn't look like a room to
me..."));
  return;
endif
player.location.window_target = newtarget;
player:tell("Ok. You are now watching ", newtarget.name, " (",
newtarget, ").");
"Creeper fecit on Mon Sep 15 20:48:16 1997 EST";
.

@verb #434:"tree_growth" this none this "rxd"
@prog #434:tree_growth
"Called by the trees details. Returns a message with the increased
height of the trees.";
height = this.tree_height = this.tree_height + 1;
if (height > 10000)
  height = this.tree_height = 1;
endif
player:tell("You see a group of trees about ", height, "cm tall.");
return;
.

@verb #434:"help_msg" this none this "rxd"
@prog #434:help_msg
"In progress. To be rewritten when the prototpyes become unified.";
return;
"Last modified Fri May 22 16:47:23 1998 EST by Creeper
(#101@Virtual_Campus).";
.

@verb #434:"init_for_core" this none this "rx"
@prog #434:init_for_core
"Copied from The System Object (#0):init_for_core by Administrator (#2)
Fri Jun  5 15:35:08 1998 EST";
if (caller_perms().wizard)
  pass();
  if ("server_started" in verbs(this))
    code = {"callers() || ($last_restart_time = time());"};
    set_verb_code(this, "server_started", code);
  endif
  if ("name_lookup_failed" in verbs(this))
    delete_verb(this, "name_lookup_failed");
  endif
  if ("uptime_since" in verbs(this))
    delete_verb(this, "uptime_since");
  endif
  if ("do_command" in verbs(this))
    delete_verb(this, "do_command");
  endif
```

```
  $shutdown_message = "";
  $shutdown_time = 0;
  $dump_interval = 3600;
  $gripe_recipients = {player};
  $class_registry = {{"generics", "Generic objects intended for use as
the parents of new objects", {$room, $exit, $thing, $note, $letter,
$container, $root_class, $player, $prog, $wiz, $generic_editor,
$mail_recipient, $mail_agent}}, {"utilities", "Objects holding useful
general-purpose verbs", {$string_utils, $gender_utils, $trig_utils,
$time_utils, $match_utils, $object_utils, $lock_utils, $list_utils,
$command_utils, $code_utils, $perm_utils, $building_utils}}};
  set_verb_code(this, "user_connected", verb_code(this,
"user_connected(core)"));
  delete_verb(this, "user_connected(core)");
endif
.

----- End of @archive #434
```

$social

```
@archive #213 with verbs and properties
----- Beginning of @archive #213

@create #206 named Social Area Prototype
;#213.("help_msg") = {"This room can be extended to be filled with
atmosphere, in a literal sense."}
;#213.("help") = {#214, #209, #207, #205}
;#213.("entrances") = {#289, #690}
;#213.("blessed_object") = #-1
;#213.("blessed_task") = 619805556
;#213.("exits") = {#527, #236, #208}
;#213.("key") = 0
;#213.("aliases") = {"Social Area Prototype"}
;#213.("description") = {"This is the prototype for a social area,
where people meet, chat and gather, for no reasons.", "Type
'@instruction here' to read the instructions on how to use this room."}
;#213.("object_size") = {45733, 896601613}
;#213.("web_calls") = 12
```

```
;#213.("about_text") = {"The Generic Lively Room (#213) offers a number
of options that allow users to create specific atmospheres within them.
These `atmospheres' are referred to as `ambient noise' on another room
generic with similar functionality, which pretty much says it all.  An
atmosphere, according to this interpretation, consists of ambient
messages that are displayed in the room.  In the case of this room
generic the fashion in which this happens and how it is triggered, can
be controlled to a great extend.", "", "Beyond that, it offers
customization of the messages that get displayed when a user connects
or disconnects or when disconnected user gets `carried off', introduces
new messages which can be displayed to users entering, leaving,
connecting or disconnecting.  The room owner(s) may also set the delay
after which the @enter or @exit messages are displayed to a user or the
delay after which a disconnected user gets carried off.", "", "Most of
the rooms functionality roots back to three generics:  the Generic
Improved Room (#184), the Generic Improved Room with Bells and Whistles
(#206) and the Generic Self Cleaning Room (#208).  Please consult
`@about #184', `@about #206', and `@about #208' to find out more about
said generics.", "", "For more information read `help #213' and
documentation referred to within it."}
;#213.("last_modified") = 896405392
@prop #213."on2" 0 "r"
@prop #213."active" 0 "r"
@prop #213."queue2" #215 "r"
@prop #213."mode" 1 "r"
@prop #213."triggers" {"enterfunc", "confunc"} "r"
@prop #213."random_range" {30, 300} "r"
@prop #213."random_messages" {} "rc"
@prop #213."random_counters" {} "r"
@prop #213."scripts" {} "r"
@prop #213."script_counters" {} "r"
@prop #213."enter_msg_delay" 10 "r"
@prop #213."exit_msg_delay" 10 "r"
@prop #213."sleeper_carry_off_delay" 300 "r"
@prop #213."enter_msg" "" "rc"
@prop #213."exit_msg" "" "rc"
@prop #213."connect_msg" "" "rc"
@prop #213."oconnect_msg" "" "rc"
@prop #213."disconnect_msg" "" "rc"
@prop #213."odisconnect_msg" "" "rc"
@prop #213."sleeper_carry_off_msg" "" "rc"
@prop #213."sleeper_drop_off_msg" "" "rc"
@prop #213."fixed_script" "" "r"
@prop #213."no_say_msg" "You struggle to speak but your words are lost
in the hushed silence of the room." "rc"
@prop #213."no_emote_msg" "You struggle to move but a strong inner
resistance keeps you still." "rc"
@prop #213."no_commun_msg" "Your efforts to express yourself to others
are futile in this hushed silence." "rc"
@prop #213."door_closed_msg" "Sorry, the door to %l is closed." "rc"
@prop #213."oopen_msg" "%n opens the door to %l." "rc"
@prop #213."open_msg" "You open the door." "rc"
@prop #213."oclose_msg" "%n closes the door to %l." "rc"
@prop #213."close_msg" "You close the door." "rc"
@prop #213."session" 0 "r"
@prop #213."lying_around_msg" "You see %stuff." "rc"
@prop #213."attach_hush" 0 "r"
```

```
@prop #213."furniture_msg" "You see %stuff." "rc"

@verb #213:"start" this none this "rx"
@prog #213:start
if (!$perm_utils:controls(caller_perms(), this))
  return E_PERM;
endif
if (!this.active && this.on2)
  fork (0)
    this:announce_atmosphere();
    this.queue2:add(this);
  endfork
  this.active = 1;
endif
"Last modified Mon May  6 21:39:38 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"stop" this none this "rx"
@prog #213:stop
if (!$perm_utils:controls(caller_perms(), this))
  return E_PERM;
endif
if (task = $list_utils:assoc(this, this.queue2.queue))
  fork (0)
    this.queue2:remove(this);
  endfork
endif
this.active = 0;
"Last modified Sun Sep 24 14:59:14 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"keep_going" this none this "rxd"
@prog #213:keep_going
return this.active ? this.active =
$set_utils:intersection(this.contents, connected_players()) ? 1 | 0 |
0;
"Last modified Sun Sep 24 14:59:23 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"lively_delay" this none this "rxd"
@prog #213:lively_delay
mode = this.mode;
if (mode in {1, 2, 3, 4})
  return this.random_range[1] + random(this.random_range[2] -
this.random_range[1]);
elseif (mode == 5)
  lens = length(this.scripts);
  if (!this.script_counters || length(this.script_counters) < 2 ||
!this.script_counters[2])
    this.script_counters = {this.scripts[i = random(lens)][1],
$list_utils:range(1, length(this.scripts[i][2]))};
  else
    i = $list_utils:iassoc(this.script_counters[1], this.scripts);
```

```
  endif
  delay = this.scripts[i][2][this.script_counters[2][1]];
  if (what = tonum(delay))
    this.script_counters[2] = listdelete(this.script_counters[2], 1);
  endif
  return what;
elseif (mode == 6)
  if (!this.script_counters)
    this.script_counters = {this.scripts[this.fixed_script || 1],
$list_utils:range(1, length(this.scripts[this.fixed_script || 1][2]))};
  elseif (!this.script_counters[2])
    i = $list_utils:iassoc(this.script_counters[1], this.scripts);
    this.script_counters[2] = $list_utils:range(1,
length(this.scripts[i][2]));
  endif
  delay = this.scripts[this.fixed_script ||
1][2][this.script_counters[2][1]];
  if (what = tonum(delay))
    this.script_counters[2] = listdelete(this.script_counters[2], 1);
  endif
  return what;
endif
"Last modified Mon May  6 22:32:05 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"announce_atmosphere" this none this "rxd"
@prog #213:announce_atmosphere
if (!$perm_utils:controls(caller_perms(), this))
  return E_PERM;
endif
if (!$string_utils:is_numeric(what = this:retrieve_msg()))
  this:announce_all(what);
endif
"Last modified Mon May  6 21:39:30 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"retrieve_msg" this none this "rx"
@prog #213:retrieve_msg
mode = this.mode;
if (mode == 1)
  if (!this.random_messages)
    msg = "Notice:  %T is set to mode 1, eventhough there are no
random-messages available.  Type `help random-messages' for more
information.";
  else
    msg = this.random_messages[random(length(this.random_messages))];
  endif
elseif (mode == 2)
  if (!this.random_messages)
    msg = "Notice:  %T is set to mode 2, eventhough there are no
random-messages available.  Type `help random-messages' for more
information.";
  else
    if (!this.random_counters)
```

```
      this.random_counters = $list_utils:range(1, len =
length(this.random_messages));
    else
      len = length(this.random_counters);
    endif
    msg = this.random_messages[this.random_counters[i = random(len)]];
    this.random_counters = listdelete(this.random_counters, i);
  endif
elseif (mode == 3)
  if (!this.scripts)
    msg = "Notice:  %T is set to random^2 script mode (mode 3),
eventhough there are no scripts available.  Type `help lively-index'
for more information.";
  else
    lens = length(this.scripts);
    i = random(lens);
    if (len = length(this.scripts[i][2]))
      msg = this.scripts[i][2][random(len)];
    else
      msg = tostr("Notice:  %T is set to random^2 script mode (mode 3);
anyhow, the ", this.scripts[i][1], " script seems to be empty.  Type
`help lively-index' for more information.");
    endif
  endif
elseif (mode == 4)
  if (!this.scripts)
    msg = "Notice:  %T is set to random script mode (mode 4),
eventhough there are no scripts available.  Type `help lively-index'
for more information.";
  else
    lens = length(this.scripts);
    if (!this.script_counters || !this.script_counters[2])
      this.script_counters = {this.scripts[x = random(lens)][1],
$list_utils:range(1, length(this.scripts[x][2]))};
      i = x;
    else
      i = $list_utils:iassoc(this.script_counters[1], this.scripts);
    endif
    if (len = length(this.script_counters[2]))
      msg = this.scripts[i][2][this.script_counters[2][j =
random(len)]];
      this.script_counters[2] = listdelete(this.script_counters[2], j);
    else
      msg = tostr("Notice:  %T is set to random script mode (mode 4);
anyhow, the ", this.scripts[i][1], " script seems to be empty.  Type
`help lively-index' for more information.");
    endif
  endif
elseif (mode == 5)
  if (!this.scripts)
    msg = "Notice:  %T is set to random sequential script mode (mode
5), eventhough there are no scripts available.  Type `help lively-
index' for more information.";
  else
    lens = length(this.scripts);
```

```
    if (!this.script_counters || length(this.script_counters) < 2 ||
!this.script_counters[2])
      this.script_counters = {this.scripts[x = random(lens)][1],
$list_utils:range(1, length(this.scripts[x][2]))};
      i = x;
    else
      i = $list_utils:iassoc(this.script_counters[1], this.scripts);
    endif
    msg = this.scripts[i][2][this.script_counters[2][1]];
    if (!$string_utils:is_numeric(msg))
      this.script_counters[2] = listdelete(this.script_counters[2], 1);
    endif
  endif
elseif (mode == 6)
  if (!this.scripts)
    msg = "Notice:  %T is set to fixed sequential script mode (mode 6),
eventhough there are no scripts available.  Type `help lively-index'
for more information.";
  else
    if (!this.script_counters)
      this.script_counters = {this.scripts[this.fixed_script || 1][1],
$list_utils:range(1, length(this.scripts[this.fixed_script || 1][2]))};
    elseif (length(this.script_counters) < 2 ||
!this.script_counters[2])
      this.script_counters[2] = $list_utils:range(1,
length(this.scripts[this.script_counters[1]][2]));
    endif
    msg = this.scripts[this.fixed_script ||
1][2][this.script_counters[2][1]];
    if (!$string_utils:is_numeric(msg))
      this.script_counters[2] = listdelete(this.script_counters[2], 1);
    endif
  endif
endif
return $string_utils:pronoun_sub(msg);
"Last modified Mon May  6 22:26:33 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"enterfunc" this none this "rxd"
@prog #213:enterfunc
pass(@args);
if (is_player(who = args[1]))
  if (msg = this:enter_msg())
    fork (this.enter_msg_delay)
      if (who.location == this)
        who:tell(msg);
      endif
    endfork
  endif
  if ($list_utils:find_prefix(verb, this.triggers))
    this:start();
  endif
endif
"Last modified Sun Oct 29 13:04:31 1995 EST by Ulf (#238@DU_Moo-
East).";
```

```
.

@verb #213:"exitfunc" this none this "rxd"
@prog #213:exitfunc
pass(@args);
if (is_player(who = args[1]))
  if (msg = this:exit_msg())
    fork (this.exit_msg_delay)
      if (who.location != this)
        who:tell(msg);
      endif
    endfork
  endif
  if (!this:keep_going())
    this:stop();
  endif
endif
"Last modified Sun Oct 29 13:04:41 1995 EST by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"confunc" this none this "rxd"
@prog #213:confunc
pass(@args);
if ($list_utils:find_prefix(verb, this.triggers))
  this:start();
endif
"Last modified Sun Sep 24 15:00:58 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"announce_connection" this none this "rxd"
@prog #213:announce_connection
this:announce(this:oconnect_msg() || tostr(player.name, " has
connected."));
if (msg = this:connect_msg())
  player:tell(msg);
endif
"Last modified Sun Sep 24 15:01:09 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"disfunc" this none this "rxd"
@prog #213:disfunc
if (!$recycler:valid(player.home) || !$object_utils:isa(player.home,
$room))
  player.home = $player_start;
endif
fork (delay = this.sleeper_carry_off_delay)
  if (valid(player) && !(player in connected_players()) &&
player.last_connect_time + delay < time() && this != player.home &&
player.location != player.home)
    fork (0)
      "forked to  protected from errors in the player's :moveto verb.";
      if (player.location != player.home)
```

```
        move(player, $limbo);
      endif
    endfork
    start = player.location;
    player:moveto(player.home);
    if (player.location != start && start.announce_clear)
      start:announce(this:sleeper_carry_off_msg() || tostr("Assistants
of the local psychology institute arrive to cart ", player.name, " off
to their dream-research labs."));
    endif
    if (player.location == player.home &&
player.location.announce_clear)
      player.home:announce(this:sleeper_drop_off_msg() || tostr("Some
assistants pass by to drop off a sleeping ", player.name, ", who just
took part in some revealing REM-phase experiments."));
    endif
  endif
endfork
this:announce(this:odisconnect_msg() || tostr(player.name, " has
disconnected."));
if (msg = this:disconnect_msg())
  player:tell(msg);
endif
if (!this:keep_going())
  this:stop();
endif
"Last modified Sun Sep 24 15:01:21 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"enter_msg exit_msg connect_msg oconnect_msg disconnect_msg
odisconnect_msg sleeper_carry_off_msg sleeper_drop_off_msg" this none
this "rx"
@prog #213:enter_msg
return $string_utils:pronoun_sub(this.(verb));
"Last modified Sun Jan 28 22:37:10 1996 EST by Ulf (#238@DU_MainMOO).";
.

@verb #213:"get_section" this none this "rxd"
@prog #213:get_section
what = this:parse_section(args[1]);
result = E_INVARG;
if (typeof(what) == ERR)
  player:tell("Error:  `", what, "'");
  player:tell("Usage:  @edit detail");
  player:tell("        @edit message on|for detail");
  player:tell("        @edit here|detail[|message]");
  player:tell("        @edit script");
  return result;
elseif (typeof(what) == LIST)
  d = this:match_detail(what[1]);
  if (d == $failed_match)
    $command_utils:object_match_failed(d, what[1]);
  elseif (d == $ambiguous_match)
    player:tell("I don't know which \"", what[1], "\" you mean.");
```

```
  else
    match = what[2] + "_dmsgs";
    if (!(index = match in $object_utils:all_properties(this)))
      player:tell("There is no @", what[2], " detail-message to be
set.");
      player:tell("Type \"@messages ", this.details[d][1], " all\" to
see the complete list of messages.");
    else
      result = this.(match)[d] || "";
    endif
  endif
else
  if (script = $list_utils:assoc(what, this.scripts))
    result = script[2];
  else
    player:tell("Error: ", E_INVARG);
  endif
endif
return result;
"Last modified Thu Apr 25 16:16:00 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"set_section" this none this "rxd"
@prog #213:set_section
if (!(caller in {$section_editor, this}) ||
!$perm_utils:controls(player, this))
  return E_PERM;
endif
what = this:parse_section(args[1]);
if (typeof(what) == LIST)
  source = args[2];
  d = this:match_detail(what[1]);
  if (d == $failed_match)
    player:tell("Oops, the detail you are editing is gone all of a
sudden!");
  elseif (d == $ambiguous_match)
    player:tell("Oops, the detail you are editing suddenly produces an
ambiguous match!  I don't know which \"", what[1], "\" you mean.");
  else
    match = what[2] + "_dmsgs";
    if (!(index = match in $object_utils:all_properties(this)))
      player:tell("Oops, there is no @", what[2], " detail-message to
be set.");
      player:tell("Type \"@messages ", this.details[d][1], " all\" to
see the complete list of messages.");
    else
      this.(match)[d] = source;
    endif
  endif
else
  if (index = $list_utils:iassoc(what, this.scripts))
    this.scripts[index][2] = args[2];
  else
    player:tell("Error: ", E_INVARG);
  endif
```

```
endif
this.last_modified = time();
"Last modified Thu Apr 25 17:34:20 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"section_msg" this none this "rxd"
@prog #213:section_msg
what = this:parse_section(args[1]);
if (typeof(what) == LIST)
  if (what[2] == "desc")
    return tostr("the description of the `", what[1], "' detail");
  else
    return tostr("the @", what[2], " message of the `", what[1], "'
detail");
  endif
elseif (typeof(what) == STR)
  return tostr("the `", what, "' script");
else
  return "???";
endif
"Last modified Thu Apr 25 17:41:46 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"@atmo*sphere" any any any "rd"
@prog #213:@atmosphere
"Usage:  @atmo*sphere [on|off]";
"        Turns on/off room-atmosphere.";
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
endif
com = $string_utils:trim(argstr);
if (com)
  if (com in {"on", "yes", "1"})
    if (this.on2)
      return player:tell("The atmosphere for ", this.name, " is already
activated.");
    endif
    this.on2 = 1;
    this:start();
  elseif (com in {"off", "no", "0"})
    if (!this.on2)
      return player:tell("The atmosphere for ", this.name, " is already
deactivated.");
    endif
    this.on2 = 0;
    this:stop();
  else
    player:tell(E_INVARG, ";  Usage:  @atmo*sphere [on|off]");
  endif
endif
player:tell("The atmosphere for ", this.name, " is ", tostr(args ? "now
" | ""), "turned ", this.on2 ? "on." | "off.");
if (this.on2 && !args)
```

```
  when = 0;
  if (entry = $list_utils:assoc(this, this.queue2.queue))
    when = entry[2];
  endif
  if (when)
    left = when - time();
    player:tell("Next event in ", $string_utils:from_seconds(left), "
(", ctime(when)[12..19], " ", ctime(when)[26..28], ")");
  else
    player:tell("There seems to be no next event scheduled,
currently.");
  endif
endif
"Last modified Sun Sep 24 16:08:08 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"@add-script" any to this "rd"
@prog #213:@add-script
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
endif
script = dobjstr;
if (!script)
  return player:tell("Usage:  ", verb, " <script-name> to ", iobjstr);
elseif ($list_utils:assoc(script, this.scripts))
  return player:tell("There already seems to be a script by that name
(", script, "), sorry.");
endif
this.scripts = listappend(this.scripts, {script, {}});
player:tell("You add a script called `", script, "' to ", this.name,
".");
if ($command_utils:yes_or_no("Would you like to edit this script
now?"))
  $section_editor:invoke(tostr(tostr(this), "||", script), "@edit");
else
  player:tell("Done.");
endif
"Last modified Thu Apr 25 17:58:50 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"@remove-script" any from this "rd"
@prog #213:@remove-script
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
endif
script = dobjstr;
if (!script)
  return player:tell("Usage:  ", verb, " <script-name> from ",
iobjstr);
elseif (!(index = $list_utils:iassoc(script, this.scripts)))
  return player:tell("There doesn't seem to be a script by that name
(", script, "), sorry.");
```

```
elseif (!$command_utils:yes_or_no(tostr("Are you sure you want to
remove `", script, "'?")))
  return player:tell("OK, aborted.");
endif
this.scripts = listdelete(this.scripts, index);
player:tell("You delete the script called `", script, "' from ",
this.name, ".");
"Last modified Mon Sep 25 14:57:23 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"@scripts" this none none "rd"
@prog #213:@scripts
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
endif
if (!this.scripts)
  player:tell("No scripts for ", $string_utils:nn(this), ".");
else
  scripts = {};
  for script in (this.scripts)
    scripts = listappend(scripts, tostr(script[1], " [",
length(script[2]), " lines]"));
  endfor
  player:tell("The scripts for ", $string_utils:nn(this), ":  ",
$string_utils:english_list(scripts), ".");
endif
"Last modified Mon May  6 18:00:48 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"@intervals @pauses" this any any "rx"
@prog #213:@intervals
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
elseif (!(prepstr in {"", "from"}) || (prepstr && !iobjstr))
  player:tell("Usage:  ", verb, " ", dobjstr, " from <min> to <max>");
elseif (!prepstr && !iobjstr)
  range = this.random_range;
  player:tell("Random pauses range from ", range[1], " to ", range[2],
" seconds.");
else
  range = $string_utils:words(iobjstr);
  if ((len = length(range)) < 2)
    if (i = index(range[1], "-") && (len2 = length(range[1])) > i && i
!= 1)
      from = range[1][1..i - 1];
      to = range[1][i + 1..len2];
    elseif (i = index(range[1], "..") && (len2 = length(range[1])) > i
+ 1 && i != 1)
      from = range[1][1..i - 1];
      to = range[1][i + 2..len2];
    else
```

```
      return player:tell("Usage:  ", verb, " ", dobjstr, " from <min>
to <max>");
    endif
  elseif (len == 3 && range[2] == "to")
    from = range[1];
    to = range[3];
  endif
  if (!$string_utils:is_numeric(from) || !$string_utils:is_numeric(to))
    return player:tell("Usage:  ", verb, " ", dobjstr, " from <min> to
<max>");
  endif
  from = tonum(from);
  to = tonum(to);
  if (from > to)
    player:tell(E_INVARG, ";  from ", from, " to ", to, "?  (backwards
range)");
  endif
  this.random_range = {from, to};
  player:tell("Random pauses will now range from ", from, " to ", to, "
seconds.");
endif
"Last modified Tue Sep 26 16:44:05 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"@activity*-modes" this is any "rd"
@prog #213:@activity-modes
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
elseif (!iobjstr)
  mode = this.mode;
  player:tell($string_utils:nn(this), " is currently set to display ",
mode == 1 ? "random pauses and random messages" | (mode == 2 ? "random
pauses and randomly sequenced messages" | (mode == 3 ? "random pauses
and random messages picked from scripts" | (mode == 4 ? "randomly
picked scripts with random sequence and timing" | (mode == 5 ?
"randomly picked scripts with script-extracted timing" | "fixed scripts
with script-extracted timing")))), " (mode ", mode, ").");
  player:tell();
  player:tell("The following activity-modes are currently available:");
  player:tell();
  player:tell("  mode 1:  random pauses and random messages");
  player:tell("  mode 2:  random pauses and randomly sequenced
messages");
  if (this.scripts)
    player:tell("  mode 3:  random pauses and random messages picked
from scripts");
    player:tell("  mode 4:  randomly picked scripts with random
sequence and timing");
    player:tell("  mode 5:  randomly picked scripts with script-
extracted sequence and timing");
    player:tell("  mode 6:  fixed scripts with script-extracted
sequence and timing");
  endif
  player:tell();
```

```
elseif (!$string_utils:is_numeric(iobjstr))
  player:tell("Usage:  ", verb, " ", dobjstr, " is <mode>");
elseif (!((mode = tonum(iobjstr)) in {1, 2, 3, 4, 5, 6}))
  player:tell("Usage:  ", verb, " ", dobjstr, " is <mode>");
  player:tell("        modes 1, 2, 3, 4, 5, and 6 are available.");
elseif (this.mode == mode)
  player:tell($string_utils:nn(this), " is already in mode ", mode,
".");
else
  this.mode = mode;
  player:tell("You set ", $string_utils:nn(this), " to display ", mode
== 1 ? "random pauses and random messages" | (mode == 2 ? "random
pauses and randomly sequenced messages" | (mode == 3 ? "random pauses
and random messages picked from scripts" | (mode == 4 ? "randomly
picked scripts with random sequence and timing" | (mode == 5 ?
"randomly picked scripts with script-extracted sequence and timing" |
"fixed scripts with script-extracted sequence and timing")))), ".");
endif
"Last modified Tue May  7 19:39:31 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"@trigger*s" this is any "rd"
@prog #213:@triggers
"Usage:  @trigger here is <trigger>";
"        @trigger here is !<trigger>";
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
endif
if (!iobjstr)
  triggers = this.triggers;
  source = {{"enterfunc", "people entering"}, {"confunc", "people
connecting into"}};
  source2 = source;
  if (triggers)
    for t in (triggers)
      player:tell($string_utils:nn(this), " gets triggered by ", (s =
$list_utils:assoc(t, source))[2], " it (", s[1], ").");
      source2 = setremove(source2, s);
    endfor
  endif
  if (source2)
    player:tell();
    player:tell("  Other triggers:");
    player:tell();
    for s in (source2)
      player:tell("  `", s[1], "':  ", s[2], " the room.");
    endfor
    player:tell();
  endif
else
  triggers = $string_utils:words(iobjstr);
  for t in (triggers)
    rm = 0;
    if (t[1] in {"!", "-"})
```

```
      if ((len = length(t)) < 2)
        player:tell("`", t, "' - ?");
      else
        t = t[2..len];
        rm = 1;
      endif
    endif
    if (index(t, "enter") || index(t, "con"))
      what = index(t, "enter") ? "enterfunc" | "confunc";
      if (rm)
        if (what in this.triggers)
          this.triggers = setremove(this.triggers, what);
          player:tell("You remove the `", what, "' atmosphere
trigger.");
        else
          player:tell("`", what, "' is not a trigger currently.");
        endif
      else
        if (what in this.triggers)
          player:tell("`", what, "' is already a trigger.");
        else
          this.triggers = setadd(this.triggers, what);
          player:tell("You add `", what, "' as an atmosphere
trigger.");
        endif
      endif
    else
      player:tell("`", t, "' - ?");
    endif
  endfor
endif
"Last modified Wed Sep 27 13:01:09 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"@fixed-script" this is any "rd"
@prog #213:@fixed-script
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
endif
if (!iobjstr)
  player:tell(tostr(this.fixed_script ? tostr("The current fixed script
for ", $string_utils:nn(this), " is: ",
this.scripts[this.fixed_script][1], ".") |
tostr($string_utils:nn(this), " has no current fixed script",
tostr(this.scripts ? "" | " and no scripts"), " defined.")));
else
  if (!(script = $list_utils:iassoc(iobjstr, this.scripts)))
    player:tell($string_utils:nn(this), " doesn't have a script by that
name (", iobjstr, ").");
  else
    if (this.fixed_script == script)
      player:tell("`", iobjstr, "' is alread the fixed script for ",
$string_utils:nn(this), ".");
```

```
    else
      this.fixed_script = script;
      player:tell("You set the fixed script of ",
$string_utils:nn(this), " to `", iobjstr, "'.");
    endif
  endif
endif
"Last modified Tue May  7 22:16:39 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"@enter-delay @exit-delay @carry-off-delay" this is any "rd"
@prog #213:@enter-delay
if (!$perm_utils:controls(player, this))
  return player:tell(this:no_perms_msg() || "You may not do that here,
sorry.");
endif
what = tostr(tostr(verb == "@enter-delay" ? "enter_msg_" | (verb ==
"@exit-delay" ? "exit_msg_" | "sleeper_carry_off_")), "delay");
if (!iobjstr)
  player:tell(tostr(what == "enter_msg_delay" ? "@enter display delay:
" | (what == "exit_msg_delay" ? "@exit display delay: " | "Sleeper
carry off delay: ")), this.(what), " seconds.");
else
  if (!$string_utils:is_numeric(iobjstr))
    player:tell("Usage:  ", verb, " ", dobjstr, " is <number of
seconds>");
  else
    this.(what) = tonum(iobjstr);
    player:tell("You set the ", tostr(what == "enter_msg_delay" ?
"@enter display delay " | (what == "exit_msg_delay" ? "@exit display
delay " | "sleeper carry off delay ")), " to ", iobjstr, " seconds.");
  endif
endif
"Last modified Wed Sep 27 13:34:03 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"verbs_for" this none this "rxd"
@prog #213:verbs_for
"Expected arguments:  <user> [OBJ], optional verb-location [OBJ]";
"Values returned:     list of verbs [LIST]";
"";
"Function:  returns verbs that <user> has access to.";
who = args ? args[1] | player;
if (!$perm_utils:controls(who, this))
  return {@pass(@args), @this.data:user_verbs($code_utils:verb_loc())};
else
  su = $string_utils;
  loc = $code_utils:verb_loc();
  verbs = {};
  for v in (verbs(loc))
    $command_utils:suspend_if_needed(0);
    v = su:words(su:strip_chars(v, "*"))[1];
    if (verb_args(loc, v) != {"this", "none", "this"})
      verbs = {@verbs, v};
```

```
    endif
  endfor
  return {@pass(@args), @verbs};
endif
"Last modified Thu Oct 12 13:54:43 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"chparent_to" this none this "rxd"
@prog #213:chparent_to
if (!$perm_utils:controls(caller_perms(), this))
  return E_PERM;
endif
if (!args[3])
  return;
endif
data = (rvalue = pass(@args))[1];
rdata = data;
log = rvalue[2];
known_parents = $list_utils:slice(this.data.conversion_tables, 2);
if ((loc = $code_utils:verb_loc()) in known_parents)
  for i in [1..length(known_parents)]
    if (known_parents[i] == loc)
      $command_utils:suspend_if_needed(0);
      table = this.data.conversion_tables[i][3];
      for item in (data)
        $command_utils:suspend_if_needed(0);
        if (item[1] in table[1])
          rdata = setremove(rdata, item);
          log = {@log, tostr("LOST:  `", item[1], "' (value: ",
$string_utils:print(what.(item[1])), ")")};
        elseif (item[1] in table[2])
          this.(item[1]) = item[2];
          rdata = setremove(rdata, item);
          log = {@log, tostr("CONV(1):  `", item[1], "'")};
        elseif (found = $list_utils:assoc(item[1], table[3]))
          this.(found[2]) = item[2];
          data = setremove(rdata, item);
          log = {@log, tostr("CONV(2):  `", item[1], "'")};
        elseif (item[1] in table[4])
        endif
      endfor
      if (log != rvalue[2])
        player:tell("Conversion of ",
$string_utils:nn(this.data.conversion_tables[i][1]), " data");
        player:tell("           to ",
$string_utils:nn(this.data.conversion_tables[i][2]), " data
completed.");
      endif
    endif
  endfor
endif
if (caller == this)
  return {rdata, log};
```

```
else
  this.data.conversion_logs = {{this, time(), player, log},
@this.data.conversion_logs};
  player:tell("Room conversion completed.");
endif
"Last modified Tue Oct 17 11:06:18 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"edit_huh" this none this "rxd"
@prog #213:edit_huh
if ((result = pass(@args)) == E_INVARG)
  what = $string_utils:words(args[1]);
  if (what && $list_utils:assoc(key = what[1], this.scripts))
  elseif (what && (len = length(what[1])) > 7 && what[1][len - 6..len]
== "-script")
    key = what[1][1..len - 7];
  else
    key = "";
  endif
  result = key ? {$section_editor, tostr("here", "||", key)} | result;
endif
return result;
"Last modified Fri May 10 12:01:08 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"parse_section" this none this "rxd"
@prog #213:parse_section
if ((result = pass(@args)) == E_INVARG)
  what = args[1];
  len = length(what);
  if (what[1] == "|" && len > 1)
    result = what[2..len];
  endif
endif
return result;
"Last modified Thu Apr 25 16:15:40 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #213:"init_for_core" this none this "rx"
@prog #213:init_for_core
"Copied from Social Area Prototype (#631):init_for_core by
Administrator (#2) Fri May 29 09:04:30 1998 EST";
"Copied from The System Object (#0):init_for_core by Administrator (#2)
Thu Sep 25 09:57:10 1997 EST";
if (caller_perms().wizard)
  pass();
  if ("server_started" in verbs(this))
    code = {"callers() || ($last_restart_time = time());"};
    set_verb_code(this, "server_started", code);
  endif
  if ("name_lookup_failed" in verbs(this))
    delete_verb(this, "name_lookup_failed");
  endif
```

```
  if ("uptime_since" in verbs(this))
    delete_verb(this, "uptime_since");
  endif
  if ("do_command" in verbs(this))
    delete_verb(this, "do_command");
  endif
  $shutdown_message = "";
  $shutdown_time = 0;
  $dump_interval = 3600;
  $gripe_recipients = {player};
  $class_registry = {{"generics", "Generic objects intended for use as
the parents of new objects", {$room, $exit, $thing, $note, $letter,
$container, $root_class, $player, $prog, $wiz, $generic_editor,
$mail_recipient, $mail_agent}}, {"utilities", "Objects holding useful
general-purpose verbs", {$string_utils, $gender_utils, $trig_utils,
$time_utils, $match_utils, $object_utils, $lock_utils, $list_utils,
$command_utils, $code_utils, $perm_utils, $building_utils}}};
  set_verb_code(this, "user_connected", verb_code(this,
"user_connected(core)"));
  delete_verb(this, "user_connected(core)");
endif
.

@verb #213:"l*ook" any any any "rxd"
@prog #213:look
if (!argstr)
  pass(@args);
else
  if (f = this:is_furniture(dobjstr))
    player:tell(this.furniture[f][4]);
    folks = setremove(this.furniture[f][2], player);
    if (folks)
      player:tell("Seated at ", this.furniture[f][1], " you see ",
$string_utils:title_list(folks), ".");
    endif
    stuff = this.furniture[f][3];
    if (stuff)
      player:tell("On ", this.furniture[f][1], " you see ",
$string_utils:title_list(stuff), ".");
    endif
  else
    pass(@args);
  endif
endif
.

@verb #213:"@addfur*niture" any any any "rd"
@prog #213:@addfurniture
if (this:is_authorized(player))
  if (argstr)
    if (this:is_furniture(argstr))
      player:tell("Sorry, there is already something here by that
name.");
      return;
    endif
```

```
    player:tell("Please describe ", argstr, ".");
    descrip = $command_utils:read();
    s = $command_utils:yes_or_no("Is this object sittable? ");
    if (s)
      player:tell("Type in a self-message for when a person sits
down.");
      player:tell(" Example: \"You sit down at the %f and make yourself
comfortable.\" (%f will get substituted with ", argstr, ").");
      sit = $command_utils:read();
      player:tell("Type in a public message for when a person sits
down.");
      player:tell(" Example: \"%n sits down at the %f.\" (%n will get
substituted with the sitter and %f with ", argstr, ").");
      osit = $command_utils:read();
      player:tell("Type in a self-message for when a person stands
up.");
      player:tell(" Example: \"You rise from the %f and stretch.\" (%f
will get substituted with ", argstr, ").");
      stand = $command_utils:read();
      player:tell("Type in a public message for when the player stands
up.");
      player:tell(" Example: \"%n rises from the %f.\" (%n will get
substituted with the sitter and %f with ", argstr, ").");
      ostand = $command_utils:read();
      player:tell("Type in a message for the room's description when
people are sitting on ", argstr, ".");
      player:tell(" Example: \"%crowd %is sitting at %f.\" (%crowd will
get substituted with the people sitting there, %is with either \"is\"
or \"are\" and %f with ", argstr, ").");
      sitting = $command_utils:read();
      s = {sit, osit, stand, ostand, sitting};
    else
      s = {};
    endif
    this.furniture = {@this.furniture, {argstr, {}, {}, descrip, s}};
    this:announce_all(player.name, " sets up a ", argstr, " in the
room.");
  else
    player:tell("To add a new object you must give it a name:
'@addfurniture <name>'");
  endif
endif
.

@verb #213:"is_furniture" this none this "rxd"
@prog #213:is_furniture
for f in [1..length(this.furniture)]
  if (index(this.furniture[f][1], args[1]) == 1)
    return f;
  else
    words = $string_utils:words(this.furniture[f][1]);
    for word in (words)
      if (index(word, args[1]) == 1)
        return f;
      endif
```

```
    endfor
  endif
endfor
return 0;
.

@verb #213:"sit" any any any "rxd"
@prog #213:sit
if (f = this:player_loc(player))
  player:tell("You are already sitting at ", this.furniture[f][1],
".");
  return;
endif
n = 0;
if (!argstr)
  sittables = {};
  for n in [1..length(this.furniture)]
    if (this.furniture[n][5])
      sittables = {@sittables, n};
    endif
  endfor
  if (sittables)
    n = random(length(sittables));
  endif
else
  n = this:is_furniture(argstr);
endif
if (n)
  if (this.furniture[n][5])
    if (this.furniture[n][5][2])
      this:announce(this:get_seat_msg(this.furniture[n], "osit"));
    else
      this:announce(player.name, " sits down at ",
this.furniture[n][1], ".");
    endif
    if (this.furniture[n][5][1])
      player:tell(this:get_seat_msg(this.furniture[n], "sit"));
    else
      player:tell("You sit down at ", this.furniture[n][1], ".");
    endif
    this.furniture[n][2] = {@this.furniture[n][2], player};
  else
    player:tell("Sorry, that is not something you can sit on.");
  endif
  player.seat = this;
else
  pass(@args);
endif
"Last modified Fri May 15 13:22:30 1998 EST by Creeper
(#101@Virtual_Campus).";
.

@verb #213:"standup" any any any "rxd"
@prog #213:standup
```

```
if (f = this:player_loc(player))
  this.furniture[f][2] = setremove(this.furniture[f][2], player);
  if (this.furniture[f][5][4])
    this:announce(this:get_seat_msg(this.furniture[f], "ostand"));
  else
    this:announce(player.name, " rises from ", this.furniture[f][1],
".");
  endif
  if (this.furniture[f][5][3])
    player:tell(this:get_seat_msg(this.furniture[f], "stand"));
  else
    player:tell("You stand up.");
  endif
endif
if (!f && caller == player)
  pass(@args);
endif
"Last modified Tue Nov  4 14:04:01 1997 EST by Creeper
(#101@Key_Campus).";
.

@verb #213:"player_loc" this none this "rxd"
@prog #213:player_loc
"return player location: 0 for not sitting anyway, else furniture-
number";
p = args[1];
furniture = 0;
for f in [1..length(this.furniture)]
  if (args[1] in this.furniture[f][2])
    furniture = f;
  endif
endfor
return furniture;
.

@verb #213:"get_seat_msg" this none this "rxd"
@prog #213:get_seat_msg
"Return the text of a seat-message, performing substitutions.";
here = this;
f = args[1][5];
which = args[2] in {"sit", "osit", "stand", "ostand", "sitting"};
text = $string_utils:substitute(f[which], {{"%n", player.name}, {"%t",
player.name}, {"%l", here.name}, {"%f", args[1][1]}, {"%crowd",
this:crowd(@args)}, {"%is", length(args[1][2]) > 1 ||
(length(args[1][2]) == 1 && player == args[1][2][1]) ? "are" | "is"}},
1);
return $string_utils:pronoun_sub($string_utils:capitalize(text));
.

@verb #213:"crowd" this none this "rxd"
@prog #213:crowd
crowd = args[1][2];
if ((rest = setremove(crowd, player)) && player in crowd)
  if (length(rest) < 2)
    first = " and ";
```

```
  else
    first = ", ";
  endif
  text = "you" + first + $string_utils:title_list(rest);
elseif (player in crowd)
  text = "you";
else
  text = $string_utils:title_list(crowd);
endif
return text;
.

@verb #213:"is_authorized" this none this "rx"
@prog #213:is_authorized
"Copied from Social Area Prototype (#631):is_authorized by
Administrator (#2) Fri May 29 09:08:18 1998 EST";
"Copied from Generic Improved Classroom (#211):is_authorized by
Administrator (#2) Tue Sep 30 15:02:10 1997 EST";
"Expected arguments:  <user> [OBJ], <verb> [STR]";
"Values returned:     TRUE || FALSE";
"";
"Function:  decide if <user> is allowed to use <verb>.";
v = this:verb_name(args[2]);
return $perm_utils:controls(args[1], this) || args[1] in
this.authorized ? 1 | (v in this.restricted_verbs ? 0 | 1);
"Last modified Mon Jul 31 12:02:09 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #213:"look_self" this none this "rxd"
@prog #213:look_self
pass(@args);
furniture = {};
for t in (this.furniture)
  furniture = {@furniture, t[1]};
endfor
if (furniture)
  player:tell("The furniture here is: ",
$string_utils:english_list(furniture, "nothing", " and a ", ", a ",
""), ".");
endif
if (f = this:player_loc(player))
  stuff = this.furniture[f][3];
  if (stuff)
    player:tell("On ", this.furniture[f][1], " you see ",
$string_utils:title_list(stuff), ".");
  endif
endif
sitters = {};
stuff = {};
for f in (this.furniture)
  if (f[2])
    sitters = {@sitters, @f[2]};
    player:tell(this:get_seat_msg(f, "sitting"));
  endif
```

```
  if (f[3])
    stuff = {@stuff, @f[3]};
  endif
endfor
.

@verb #213:"door_closed_msg oopen_msg open_msg oclose_msg close_msg
standing_msg lying_around_msg" this none this "rxd"
@prog #213:door_closed_msg
"Return the text of a message, performing substitutions.";
here = this;
if (args)
  bunch = args[1];
else
  bunch = "";
endif
text = $string_utils:substitute(this.(verb), {{"%n", player.name},
{"%t", player.name}, {"%l", here.name}, {"%crowd", bunch}, {"%stuff",
bunch}}, 1);
return text;
.

@verb #213:"@rmfur*niture" any any any "rd"
@prog #213:@rmfurniture
if (this:is_authorized(player))
  if (!argstr)
    player:tell("To remove something you must name it:  '@rmfurniture
<name>'");
    return;
  endif
  if (found = this:is_furniture(argstr))
    this:announce_all(player.name, " removes ",
this.furniture[found][1], ".");
    this.furniture = listdelete(this.furniture, found);
  else
    player:tell("Sorry, there is no furniture here by that name.");
  endif
endif
.

----- End of @archive #213
```

$learning

```
@archive #211 with verbs and properties
----- Beginning of @archive #211

@create #206 named Learning Area Prototype,learning
;#211.("html_desc_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("details") = {{"desk4"}, {"desk5"}, {"teacher desk"}, {"desk3"},
{"desk2"}, {"desk1"}, {"teacher's desk", "desk"}, {"big table",
"table"}, {"blackboard", "board", "bb"}, {"clock"}, {"door"}}
```

```
;#211.("desc_dmsgs") = {0, 0, 0, 0, {"*call*", "clockface"}, 0, "You
see a massive, old desk made from oak.", "You see a big circular table
in the middle of the room and around it alot of chairs.", {"*call*",
"blackboard"}, {"*call*", "clockface"}, "You see a white door that is
partially covered by stickers and flyers."}
;#211.("take_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "The %t is too heavy
to move.", "You can't pick the %t up, you can hardly move it.", 0, 0,
"The %t seems well-anchored in its hinges."}
;#211.("otake_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("seats") = {2, 2, 1, 2, 2, 2, 1, 20, 0, 0, 0}
;#211.("sitting") = {}
;#211.("sitting_seats") = {}
;#211.("capacity") = {0, 0, 0, 0, 0, 0, 3, 10, 0, 0, 0}
;#211.("contained") = {}
;#211.("containing_details") = {}
;#211.("sit_dmsgs") = {0, 0, "You sit down at the %t.", 0, "You sit
down at the %t.", "You sit down at the %t.", "You sit down at the %t.",
"You sit down at the %t.", 0, 0, 0}
;#211.("osit_dmsgs") = {0, 0, "%N %<sits> at the %t.", 0, "%N %<sits>
at the %t.", "%N %<sits> at the %t.", "%N %<sits> down at the %t.", "%N
%<sits> down at the %t.", 0, 0, 0}
;#211.("sit_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "You can't sit at the
%t.", "You can't sit at the %t.", 0, 0, "You can't sit on %t."}
;#211.("osit_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "%N %<tries> to sit
down at the %t.", "%N %<tries> to sit at the %t.", 0, 0, ""}
;#211.("stand_dmsgs") = {0, 0, 0, 0, 0, 0, "You stand up from the %t.",
"You stand up from the %t.", 0, 0, 0}
;#211.("ostand_dmsgs") = {0, 0, 0, 0, 0, 0, "%N %<stands> up from the
%t.", "%N %<stands> up from the %t.", 0, 0, 0}
;#211.("sitting_dmsgs") = {0, 0, 0, 0, 0, 0, "%N %<is> sitting at the
%t.", "%N %<is> sitting at the %t.", 0, 0, 0}
;#211.("put_dmsgs") = {0, 0, 0, 0, 0, 0, "You put %d on the %t.", "You
put %d on the %t.", 0, 0, 0}
;#211.("oput_dmsgs") = {0, 0, 0, 0, 0, 0, "%N %<puts> %d on the %t.",
"%N %<puts> %d on the %t.", 0, 0, 0}
;#211.("put_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "You can't put %d on
the %t.", "You can't put %d on the %t.", 0, 0, "You can't put %d on the
%t."}
;#211.("oput_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "", "", 0, 0, ""}
;#211.("get_dmsgs") = {0, 0, 0, 0, 0, 0, "You get %d from the %t.",
"You get %d from the %t.", 0, 0, 0}
;#211.("oget_dmsgs") = {0, 0, 0, 0, 0, 0, "%N %<gets> %d from the %t.",
"%N %<gets> %d from the %t.", 0, 0, 0}
;#211.("get_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "You can't get %d from
the %t.", "You can't get %d from the %t.", 0, 0, "You can't get %d from
the %t."}
;#211.("oget_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "", "", 0, 0, ""}
;#211.("contained_dmsgs") = {0, 0, 0, 0, 0, 0, "You see %n on top of
the %t.", "You see %n on top of the %t.", 0, 0, 0}
;#211.("help_msg") = ""
;#211.("touch_dmsgs") = {0, 0, 0, 0, 0, 0, "The surface feels smooth,
yet bumpy.", "You let your hand glide across the smooth surface of the
table.", 0, 0, 0}
;#211.("otouch_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("smell_dmsgs") = {0, 0, 0, 0, 0, 0, "A scent of oak and paper is
all you can make out.", "It smells.. neutral. In fact it doesn't smell
at all.", 0, 0, 0}
;#211.("osmell_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

```
;#211.("listen_dmsgs") = {0, 0, 0, 0, 0, 0, "You hear nothing also
known as the sound of silence.", "Fortunately it doesn't produce any
sounds.", 0, 0, "On occassion the door creaks when being opened or
closed."}
;#211.("olisten_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("taste_dmsgs") = {0, 0, 0, 0, 0, 0, "If you were to actually
taste it you'd taste nothing much.", "What would you rather do? Put
your tongue on the %t or imagine what it might taste like?", 0, 0, 0}
;#211.("otaste_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("opened_containers") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}
;#211.("invisible_contents") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("open_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, {"*call*",
"open_door", "You open the %t."}}
;#211.("oopen_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "%N %<opens> the
%t."}
;#211.("close_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, {"*call*",
"close_door", "You close the %t."}}
;#211.("oclose_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "%N %<closes>
the %t."}
;#211.("open_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "You can't open the
%t, it has no drawers.", "You can't open the %t.", 0, 0, 0}
;#211.("oopen_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("close_failed_dmsgs") = {0, 0, 0, 0, 0, 0, "You can't close
that, there's nothing that could be closed.", "You can't close the
%t.", 0, 0, 0}
;#211.("oclose_failed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
;#211.("opened_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "The %t is
open."}
;#211.("closed_dmsgs") = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, "The %t is
closed."}
;#211.("editing_buffer") = {"*call*", "blackboard"}
;#211.("editing") = {3, "desc_dmsgs"}
;#211.("data") = #218
;#211.("generic") = #211
;#211.("help") = {#212, #209, #207, #205}
;#211.("exits") = {#527, #236, #315}
;#211.("instruction") = {"The Classroom.", "", "----------------------
----------------------------------------", "Please have a seat at the
table.", "(To sit at the table, type 'sit'.)", "", "While class is in
session, please stay focused and on topic:", "'look bb' to see the
current topics on the blackboard.", "If you wish to have a casual
conversation, go somewhere else.", "", "If the 'say' command does not
work, try 'speakup'", "or 'su'.  For more detailed instructions, type
'help here'.", "----------------------------------------------------
----------", ""}
;#211.("key") = 0
;#211.("aliases") = {"Learning Area Prototype", "learning"}
;#211.("description") = {"This is the prototype for the learning area.
You will find tools here which help a teacher to give lectures, or give
tutorials, or any other exchange of information with students.", "You
see several desks (desk1, desk2, desk3, desk4, desk5), all of them for
two people each.", "Then there is the teacher desk, and a blackboard
behind it.", "You can record activity in this room by selecting a note,
with @note is ..., and then turning the recording feature on, with
@record on|off.", "A clock on the wall countdowns time."}
;#211.("object_size") = {65961, 896601613}
;#211.("web_calls") = 58
```

```
;#211.("about_text") = {"The Generic Improved Classroom (#211) allows
instructors or groups to emulate classroom settings.  A classroom can
be set up to support different discourse or conversation modes;  you
could build an big auditorium which would resemble a lecture setting or
you could assemble a large group discussion area in which different
groups can have discourses within the same room simultaneously without
disturbing one another.", "", "The key to alot of this functionality
lies in the classroom's moderation facilities, which allow classroom
owners and authorized users of classrooms to control the sources of
expression within that room.  For example, an instructor might decide
s/he wants all students to use the `to' (directed speech verb) during a
class in session, but not the `emote' verb;  the classroom would allow
her to set it up accordingly.  Classrooms also allow conversations to
be localized at tables.  That means a classroom in session will cause
things said and emoted at tables to be only heard by others sitting at
that same table.", "", "Other features include a special classroom
security that can be switched by simply opening or closing a door and
that is sensitive to currently held classes (it would admit students of
a class currently set up but not other people).  Classrooms can be
shared among a number of groups and then set up for individual classes
(which in turn influences the classroom security).  There is a simple
blackboard that can be used, a clock and a default combination of a
teacher's desk and a big table for students.  This initial interior
structure can be modified, of course.", "", "Much of the classroom's
functionality roots back to other generics:  the Generic Improved Room
(#184), the Generic Improved Room with Bells and Whistles (#206), and
the Generic Improved Self Cleaning Room (#208).  Please consult `@about
#184', `@about #206', and `@about #208' to find out more about said
generics.  Also, when standing in a classroom, take a look at `help
index'.  The rooms-index, the bells-index and the self-cleaning-index
all refer to collections of help topics that relate to aforementioned
generics.  To list theses indices type `help <index>' (for example:
help room-index).", "", "For more information read `help classroom-
index' and documentation referred to within it.  Note that this help-
index is only available within classrooms."}
;#211.("last_modified") = 890864024
@prop #211."restricted_verbs" {"@restrict*ions", "@stifle",
"@auth*orized", "@invisible*-session", "@addspeaker-s*eats",
"@session", "@stat*us", "@mkclass", "@rmclass", "@class*es",
"reg*ister", "writeb*lackboard", "eraseb*lackboard",
"cleanb*lackboard", "open", "sit"} "r"
@prop #211."allowed_sources" {"announce_connection", "disfunc", "sit",
"stand", "@go", "@join", "home", "emote", "writeblackboard",
"eraseblackboard", "cleanblackboard", "open", "@session",
"announce_msg"} "r"
@prop #211."protected_details" {11, 10, 9} "r"
@prop #211."speaker_seats" {7} "r"
@prop #211."classes" {} "r"
@prop #211."authorized" {} "r"
@prop #211."session" 0 "r"
@prop #211."current_class" 0 "r"
@prop #211."hide_when_in_session" 0 "r"
@prop #211."door_open" 1 "r"
@prop #211."show_door" 1 "r"
@prop #211."numbered_bb" 1 "r"
@prop #211."blackboard" {"hello"} "rc"
@prop #211."restricted_msg" "You are not allowed to do that." "rc"
@prop #211."bb_empty_msg" "The %t appears to be freshly wiped - clean
and empty." "rc"
```

```
@prop #211."blank_write_bb_msg" "You wipe a clean line underneath the
text written on the %t." "rc"
@prop #211."write_bb_msg" "You step up the %t and scribble onto it."
"rc"
@prop #211."owrite_bb_msg" "%N steps up to the %t and scribbles
something onto it." "rc"
@prop #211."erase_bb_msg" "You step up the %t and erase a line." "rc"
@prop #211."oerase_bb_msg" "%N steps up to the %t and erases a line."
"rc"
@prop #211."clean_bb_msg" "You step up to the %t wipe it with a soapy
sponge." "rc"
@prop #211."oclean_bb_msg" "%N steps up to the %t and wipes it with a
soapy sponge." "rc"
@prop #211."hush_msg" "Shhh, the class is in session. If you wish to
speakup then please do so by typing:  su <text>" "rc"
@prop #211."hush_intruder_msg" "Shhh, %t is in session, so don't
disturb." "rc"
@prop #211."session_msg" " (In Session)" "rc"
@prop #211."session_begin_msg" "%N announces the beginning of a new
session." "rc"
@prop #211."session_end_msg" "%N announces the end of this session."
"rc"
@prop #211."developer" {"Ulf Kastner", "ulf@du.org"} "r"
@prop #211."hush_controllers" 0 "r"
@prop #211."hide_occupants" 0 "rc"

@verb #211:"blackboard" this none this "rxd"
@prog #211:blackboard
"Expected arguments:  detail-index [NUM]";
"Values returned:     blackboard text [LIST]";
"";
"Function:  returns text stored in the \"blackboard\" property.";
dtname = this.details[args[1]][1];
len = player:linelen();
borders = {$string_utils:center(tostr(" ",
$string_utils:uppercase(dtname), " "), len, "="),
$string_utils:space(len, "=")};
if (this.blackboard)
  lines = this.blackboard;
  if (this.numbered_bb)
    for i in [1..length(lines)]
      lines[i] = tostr(i, ")  ", this.blackboard[i]);
    endfor
  endif
  lines = listinsert(lines, borders[1], 1);
  lines = listappend(lines, borders[2]);
  if ((c = callers()) && "html" in $list_utils:slice(c, 2))
    lines = {"<PRE>", @lines, "</PRE>"};
  endif
else
  lines = {this.bb_empty_msg};
endif
return lines;
"Last modified Thu Nov  2 23:53:37 1995 EST by Ulf (#238@DU_Moo-
East).";
```

```
.

@verb #211:"writeb*lackboard writebb wbb" any any any "rd"
@prog #211:writeblackboard
"Usage:  writeb*lackboard <text>";
"Append <text> to the writing on the blackboard.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
else
  this.blackboard = {@this.blackboard, argstr};
  if (argstr)
    if (index = player in this.sitting)
      seat = this.details[this.sitting_seats[index]][1];
      this:stand();
    else
      seat = 0;
    endif
    player:tell(this:write_bb_msg());
    if (msg = this:owrite_bb_msg())
      this:announce(msg);
    endif
    if (seat)
      argstr = seat;
      this:sit();
    endif
  else
    player:tell(this:blank_write_bb_msg());
  endif
endif
"Last modified Sun Jul 30 22:47:27 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"eraseb*lackboard erasebb ebb" any none none "rd"
@prog #211:eraseblackboard
"Usage:  eraseb*lackboard <line-number>";
"Erase the <line-number>th line on the blackboard.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
else
  line = $code_utils:tonum(dobjstr);
  if (!typeof(line) == NUM)
    return player:tell("Usage:  ", verb, " <line-number>");
  endif
  len = length(this.blackboard);
  if (line > len)
    return player:tell("Sorry, there are only ", len, " lines on the
blackboard.");
  endif
  this.blackboard = listdelete(this.blackboard, line);
  if (index = player in this.sitting)
    seat = this.details[this.sitting_seats[index]][1];
    this:stand();
  else
```

```
    seat = 0;
  endif
  player:tell(this:erase_bb_msg());
  if (msg = this:oerase_bb_msg())
    this:announce(msg);
  endif
  if (seat)
    argstr = seat;
    this:sit();
  endif
endif
"Last modified Sat Oct  7 14:11:09 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"cleanb*lackboard cleanbb cbb" none none none "rd"
@prog #211:cleanblackboard
"Usage:  cleanb*lackboard";
"Erase all writing on the blackboard.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
else
  this.blackboard = {};
  if (index = player in this.sitting)
    seat = this.details[this.sitting_seats[index]][1];
    this:stand();
  else
    seat = 0;
  endif
  player:tell(this:clean_bb_msg());
  if (msg = this:oclean_bb_msg())
    this:announce(msg);
  endif
  if (seat)
    argstr = seat;
    this:sit();
  endif
endif
"Last modified Sun Jul 30 22:50:11 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"restricted_msg blank_write_bb_msg write_bb_msg
owrite_bb_msg erase_bb_msg oerase_bb_msg clean_bb_msg oclean_bb_msg
hush_msg hush_intruder_msg session_begin_msg session_end_msg" this none
this "rx"
@prog #211:restricted_msg
"Expected arguments:  NONE";
"Values returned:     message-text [STR]";
"";
"Function:  returns text stored in the this.(verb) property after
performing some substitutions.";
msg = this.(verb);
if (index(verb, "bb"))
```

```
  dtname = (i = this:match_detail("bb")) ? this.details[i][1] |
"blackboard";
  msg = strsub(msg, "%t", dtname, 1);
  msg = strsub(msg, "%T", $string_utils:capitalize(dtname), 1);
endif
return $string_utils:pronoun_sub(msg, player, this, this);
"Last modified Sun Jul 30 22:54:02 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"is_authorized" this none this "rx"
@prog #211:is_authorized
"Expected arguments:  <user> [OBJ], <verb> [STR]";
"Values returned:     TRUE || FALSE";
"";
"Function:  decide if <user> is allowed to use <verb>.";
v = this:verb_name(args[2]);
return $perm_utils:controls(args[1], this) || args[1] in
this.authorized ? 1 | (v in this.restricted_verbs ? 0 | 1);
"Last modified Mon Jul 31 12:02:09 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"may_speak_using" this none this "rx"
@prog #211:may_speak_using
"Expected arguments:  <user> [OBJ], <verb> [STR]";
"Values returned:     TRUE || FALSE";
"";
"Function:  decide whether <user> may speak in the room using <verb>.";
v = this:verb_name(args[2]);
return !(this.hush_controllers ? args[1] in this.sitting | 0) &&
($perm_utils:controls(args[1], this) || args[1] in this.authorized) ||
v in this.allowed_sources;
"Last modified Thu Jan 18 14:02:31 1996 EST by Ulf (#238@DU_MainMOO).";
.

@verb #211:"announce" this none this "rxd"
@prog #211:announce
"Expected arguments:  a line of text [STR]";
"Values returned:     0 <nothing>";
"";
"Function:  this is mostly an extension of $room:announce() to allow
source-based filtering with the help of may_speak_using() if the room
is in session.";
source = callers() ? callers()[1][2] | "none";
if (this.session && !this:may_speak_using(player, source))
  index = player in this.sitting;
  if (index && this.sitting_seats[index] in this.speaker_seats)
  else
    if (player in this:contents())
      player:tell(this:hush_msg());
    else
      player:tell(this:hush_intruder_msg());
    endif
    kill_task(task_id());
```

```
  endif
endif
for p in (setremove(this:contents(), player))
  p:tell(@args);
endfor
"Last modified Thu Oct 12 14:36:52 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"announce_all" this none this "rxd"
@prog #211:announce_all
"Expected arguments:  a line of text [STR]";
"Values returned:     0 <nothing>";
"";
"Function:  see announce()";
source = callers() ? callers()[1][2] | "none";
if (this.session && !this:may_speak_using(player, source))
  index = player in this.sitting;
  if (index && this.sitting_seats[index] in this.speaker_seats)
  else
    if (player in this:contents())
      player:tell(this:hush_msg());
    else
      player:tell(this:hush_intruder_msg());
    endif
    kill_task(task_id());
  endif
endif
for p in (this:contents())
  p:tell(@args);
endfor
"Last modified Thu Oct 12 11:40:10 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"announce_all_but" this none this "rxd"
@prog #211:announce_all_but
"Expected arguments:  a line of text [STR]";
"Values returned:     0 <nothing>";
"";
"Function:  see announce()";
source = callers() ? callers()[1][2] | "none";
if (this.session && !this:may_speak_using(player, source))
  index = player in this.sitting;
  if (index && this.sitting_seats[index] in this.speaker_seats)
  else
    if (player in this:contents())
      player:tell(this:hush_msg());
    else
      player:tell(this:hush_intruder_msg());
    endif
    kill_task(task_id());
  endif
endif
```

```
text = listdelete(args, 1);
contents = this:contents();
for l in (args[1])
  contents = setremove(contents, l);
endfor
for p in (contents)
  p:tell(@text);
endfor
"Last modified Thu Oct 12 11:40:23 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"announce_lines" this none this "rxd"
@prog #211:announce_lines
"Expected arguments:  text [LIST] || a line of text [STR]";
"Values returned:     0 <nothing>";
"";
"Function:  this is mostly an extension of the Generic Improved Room's
announce_lines() to allow source-based filtering with the help of
may_speak_using() if the room is in session.";
source = callers() ? callers()[1][2] | "none";
if (this.session && !this:may_speak_using(player, source))
  index = player in this.sitting;
  if (index && this.sitting_seats[index] in this.speaker_seats)
  else
    if (player in this:contents())
      player:tell(this:hush_msg());
    else
      player:tell(this:hush_intruder_msg());
    endif
    kill_task(task_id());
  endif
else
  msg = args[1];
  if (typeof(msg) == STR)
    msg = {msg};
  endif
  contents = setremove(this:contents(), player);
  for line in (msg)
    for p in (contents)
      p:tell(line);
    endfor
  endfor
endif
"Last modified Thu Aug  3 23:14:05 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"speak*up su speak_up speak-up" any any any "rd"
@prog #211:speakup
"Usage:  speak*up <text>";
"Speak up in a class that is in session and restricts the usage of
\"say\" and other common verbs of verbal expression.";
player:tell("You speak up, \"", argstr, "\"");
```

```
msg = tostr(player.name, " speaks up, \"", argstr, "\"");
for p in (setremove(this:contents(), player))
  p:tell(msg);
endfor
"Last modified Tue Aug  1 15:55:25 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"say" any any any "rxd"
@prog #211:say
"Usage:  say <text>";
"Speak to others in the classroom.  If the the classroom is in session
and \"say\" is stifled, you will only be heard by the group sitting at
a table with you.";
if (msg = $perm_utils:action_restricted(player, "speak"))
  return player:tell(msg);
endif
if (this.session && (index = player in this.sitting) && !((where =
this.sitting_seats[index]) in this.speaker_seats) &&
!this:may_speak_using(player, verb))
  seatn = this.details[where][1];
  if (length(this:detail_crowd(where)) > 1)
    player:tell("You say, \"", argstr, "\" to the others at ", seatn,
".");
    for p in (setremove(this:detail_contents(where), player))
      p:tell(player.name, " says, \"", argstr, "\" to the others at ",
seatn, ".");
    endfor
  else
    player:tell("You say, \"", argstr, "\" to yourself.");
    if (contents = setremove(this:detail_contents(where), player))
      for p in (contents)
        p:tell(player.name, " says, \"", argstr, "\" to ", player.pr,
".");
      endfor
    endif
  endif
else
  pass(@args);
endif
"Last modified Mon Oct  9 15:00:30 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"emote" any any any "rxd"
@prog #211:emote
"Usage:  emote <text>";
"Emote an action to others in the classroom.  If the classroom is in
session and general emoting is restricted it will only be seen by the
group sitting at a table with you.";
if (msg = $perm_utils:action_restricted(player, "speak"))
  return player:tell(msg);
endif
```

```
if (this.session && (index = player in this.sitting) && !((seat =
this.sitting_seats[index]) in this.speaker_seats) &&
!this:may_speak_using(player, verb))
  if (argstr != "" && argstr[1] == ":")
    for p in (this:detail_contents(seat))
      p:tell(player.name, argstr[2..length(argstr)]);
    endfor
  else
    for p in (this:detail_contents(seat))
      p:tell(player.name, " ", argstr);
    endfor
  endif
else
  pass(@args);
endif
"Last modified Thu Aug  3 23:34:28 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"to `* !*" any any any "rd"
@prog #211:to
"Usage: to <person> <text>";
"       `<person> <text>";
"       !<person> <text>";
"Direct your words to <person>.  If the classroom is in session and the
\"to\" verb is stifled, only people sitting at the table with you will
see your directed speech, that is, if <person> sits at the same table
as you; if that is not the case, the \"to\" results in direct
whispering that only <person> can hear.";
if (verb[1] in {"`", "!"})
  if (length(verb) > 1)
    name = verb[2..length(verb)];
  else
    name = argstr[1..index(argstr, " ") - 1];
  endif
  text = argstr;
else
  name = argstr[1..index(argstr, " ") - 1];
  text = argstr[index(argstr, " ") + 1..length(argstr)];
endif
who = player:my_match_object(name, player.location);
index = player in this.sitting;
if (this.session && !this:may_speak_using(player, verb))
  if (index && !(who in (cont = this:detail_contents(seat =
this.sitting_seats[index]))) && valid(who) || valid(who))
    from = index ? tostr(" from ", this.details[seat][1], ".") | "";
    who:tell(player.name, " whispers, \"", text, "\"", from);
    player:tell("You whisper, \"", text, "\" to ", who.name, ".");
  else
    who = valid(who) ? who.name | name;
    msg = tostr(player.name, " [to ", who, "]: ", text);
    for p in (cont)
      p:tell(msg);
    endfor
  endif
```

```
else
  who = valid(who) ? who.name | name;
  msg = tostr(player.name, " [to ", who, "]: ", text);
  this:announce_all(msg);
endif
"Last modified Thu Oct 12 12:43:34 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"verb_name*s list_all_verb_names" this none this "rx"
@prog #211:verb_names
"Expected arguments:  a list of verb-names [LIST] || single verb-
string";
"Values returned:     list of verb-names";
"";
"Function:  verb_name*s returns original verb name*s for provided verb-
aliases";
"                   if respective verbs are defined on the classroom
generic.";
"           list_all_verb_names returns all verb names and aliases for
all";
"                   provided verbs if respective verbs are defined on
the";
"                   classroom generic.";
verbs = typeof(args[1]) == STR ? {args[1]} | args[1];
names = {};
if (verb[1] == "v")
  for v in (verbs)
    names = {@names, (i = verb_info(this.generic, v)) != E_VERBNF ?
$string_utils:words(i[3])[1] | v};
  endfor
else
  for v in (verbs)
    v = $string_utils:strip_chars(v, "*");
    if ((i = verb_info(this.generic, v)) != E_VERBNF)
      vnames = $string_utils:words(i[3]);
      names = {@names, vnames};
    else
      names = {@names, {v}};
    endif
  endfor
endif
return length(names) == 1 ? names[1] | names;
"Last modified Thu Aug  3 23:59:24 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"verbs_for" this none this "rxd"
@prog #211:verbs_for
"Expected arguments:  <user> [OBJ], optional verb-location [OBJ]";
"Values returned:     list of verbs [LIST] || <pass>";
"";
"Function:  returns verbs that <user> has access to.";
who = args ? args[1] | player;
loc = length(args) > 1 ? args[2] | this.generic;
```

```
su = $string_utils;
verbs = {};
for v in (verbs(loc))
  $command_utils:suspend_if_needed(0);
  v = su:words(su:strip_chars(v, "*"))[1];
  if (verb_args(loc, v) != {"this", "none", "this"} &&
this:is_authorized(who, v))
    verbs = {@verbs, v};
  endif
endfor
verbs = setremove(verbs, "@rmdetail");
if (length(args) > 1)
  return {@pass(@args), @verbs};
endif
return verbs;
"Last modified Thu Oct 12 13:34:35 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"hidden_verbs" this none this "rxd"
@prog #211:hidden_verbs
"Expected arguments:  <user> [OBJ]";
"Values returned:     list of hidden verbs [LIST]";
"";
"Function:  hides verbs that <user> is not allowed to use anyway.";
su = $string_utils;
if (caller == this || $perm_utils:controls(caller_perms(), this))
  hidden = pass(@args);
  loc = $code_utils:verb_loc();
  hide = $set_utils:diff(this:verbs_for(this.owner, loc),
this:verbs_for(args[1], loc));
  for v in (hide)
    $command_utils:suspend_if_needed(0);
    vloc = $object_utils:match_verb(this, v)[1];
    hidden = setadd(hidden, {vloc, verb_info(vloc, v)[3],
verb_args(vloc, v)});
  endfor
  hidden = setadd(hidden, {generic = this.generic, verb_info(generic,
"@rmdetail")[3], verb_args(generic, "@rmdetail")});
  hidden = setadd(hidden, {loc, verb_info(loc, "@rmdetail")[3],
verb_args(loc, "@rmdetail")});
  return hidden;
else
  return E_PERM;
endif
"Last modified Fri Nov 22 10:54:32 1996 EST by BillC
(#1918@DU_MainMOO).";
.

@verb #211:"help_msg" this none this "rxd"
@prog #211:help_msg
"Expected arguments:  NONE";
"Values returned:     help-documentation [LIST]";
"";
```

```
"Function:  merges verb-documentation for interesting verbs and spews
out a resulting help text.";
verbs = this:verbs_for(player);
generic = this.generic;
su = $string_utils;
msg = {};
for v in (verbs)
  $command_utils:suspend_if_needed(0);
  if (doc = $code_utils:verb_documentation(generic, v))
    vnames = su:words(verb_info(generic, v)[3]);
    if (length(vnames) == 1 && vnames[1] == v)
      head = su:uppercase(v);
    else
      head = tostr(su:uppercase(v), "  (", su:english_list(vnames),
")");
    endif
    msg = {@msg, head, @doc, ""};
  endif
endfor
msg = {@msg, tostr("For more information, type:  @about ", generic)};
return msg;
"Last modified Thu Aug  3 23:59:28 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"detail_contents detail_crowd" this none this "rxd"
@prog #211:detail_contents
"Expected arguments:  detail-index [NUM]";
"Values returned:     objects list [LIST]";
"";
"Function:  returns contents of a given detail (this includes sitters
and contained objects)";
contents = {};
detail = args[1];
if (this.sitting_seats)
  for i in [1..length(this.sitting_seats)]
    if (this.sitting_seats[i] == detail)
      contents = listappend(contents, this.sitting[i]);
    endif
  endfor
endif
if (verb[8..12] == "crowd")
  return contents;
elseif (this.containing_details)
  for i in [1..length(this.containing_details)]
    if (this.containing_details[i] == detail)
      contents = listappend(contents, this.contained[i]);
    endif
  endfor
endif
return contents;
"Last modified Fri Aug  4 10:10:27 1995 EDT by Ulf (#238@DU_Moo-
East).";
.
```

```
@verb #211:"count_sitters" this none this "rxd"
@prog #211:count_sitters
"Expected arguments:  detail-index [NUM]";
"Values returned:     number of sitters [NUM]";
"";
"Function:  see above.";
if (args[1] in this.speaker_seats && !this:is_authorized(player,
"sit"))
  if (!$perm_utils:controls(player, this) && !(player in
this.authorized))
    return this.seats[args[1]];
  endif
endif
return pass(@args);
"Last modified Thu Aug  3 23:59:30 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"title" this none this "rxd"
@prog #211:title
"Expected arguments:  NONE";
"Values returned:     name+title string [STR]";
"";
"Function:  displays the \"title_msg\" after the room-name; when in
session it displays \"session_msg\" instead.";
if (this.session)
  return this.name + this.session_msg;
else
  bracket = this.brackets;
  return this.name + (this.title_msg ? " " + bracket[1] +
this.title_msg + bracket[2] | "");
endif
"Last modified Thu Aug  3 23:59:31 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"look_self" this none this "rx"
@prog #211:look_self
"Expected arguments:  NONE";
"Values returned:     0 <nothing>";
"";
"Function:  appends the \"@opened\" or \"@closed\" message for the door
to the room description if the \"show_door\" flag is set.";
pass(@args);
if (this.show_door)
  door = this:match_detail("door");
  if (typeof(door) != ERR)
    if (this.opened_containers[door] == -1)
      return player:tell_lines(this:closed_dmsgs(door));
    elseif (this.opened_containers[door] == 1)
      player:tell_lines(this:opened_dmsgs(door));
    endif
  endif
endif
```

```
"Last modified Tue Apr 30 13:15:40 1996 EDT by Ulf (#238@DU_MainMOO).";
.

@verb #211:"acceptable" this none this "rxd"
@prog #211:acceptable
"Expected arguments:  <user> [OBJ]";
"Values returned:     FALSE || <pass>";
"";
"Function:  enables classroom specific security.";
who = args[1];
if (this.door_open || $perm_utils:controls(who, this) || who in
this.authorized || (this.current_class && who in
this.classes[this.current_class][2]) || $list_utils:assoc(who,
this.invited) || !is_player(who))
  return pass(@args);
else
  if (who == this.blessed_object && task_id() == this.blessed_task)
    if (msg = this:walk_repel_msg())
      who:tell(msg);
    endif
    if (who != player && (msg = this:pwalk_repel_msg()))
      player:tell(msg);
    endif
    if (msg = this:owalk_repel_msg())
      this:announce_all(msg);
    endif
  endif
  return 0;
endif
"Last modified Fri Oct 13 12:36:13 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"open_door close_door" this none this "rxd"
@prog #211:open_door
"Expected arguments:  NONE";
"Values returned:     0 <nothing>";
"";
"Function:  simply toggles the \"door_open\" property.";
if (!callers() || caller != this)
  return E_PERM;
endif
if (verb[1] == "o")
  this.door_open = 1;
else
  this.door_open = 0;
endif
"Last modified Sun Oct  8 13:15:01 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"open close" any any any "rd"
@prog #211:open
"Usage:  open/close <detail>";
```

```
"Open or close a detail (or attempt to do so).  If <detail> is the
door, it may only be opened and closed if the use of \"open\" is not
restricted.  Opening or closing the door toggles room-security.  If the
door is closed only students of the currently set up class and
authorized users may enter the classroom.";
if (index(argstr, "door") && !this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
pass(@args);
"Last modified Thu Aug  3 23:59:33 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"find_class" this none this "rx"
@prog #211:find_class
"Expected arguments:  class-name [STR]";
"Values returned:     class-index [NUM] || {} [LIST] || list of class-
indeces for ambiguous result [LIST]";
"";
"Function:  tries to find a class that matches the argument. Exact
matches are returned immediately, failed matches are returned as the
empty list.";
found = {};
for c in [1..length(this.classes)]
  if (index(this.classes[c][1], argstr) == 1)
    if (this.classes[c][1] == argstr)
      return c;
    else
      found = {@found, c};
    endif
  endif
endfor
return (len = length(found)) == 1 ? found[1] | found;
"Last modified Thu Aug  3 23:59:36 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"@invisible*-session @show-d*oor @numbers @hush-
c*ontrollers" any any any "rxd"
@prog #211:@invisible-session
"Usage:  @invisible-session on|off  [A]";
"        @show-door         on|off  [B]";
"        @numbers           on|off  [C]";
"        @hush-controllers  on|off  [D]";
"If enabled, [A] the classroom will hide its occupants from @who
listings during sessions, [B] the @opened or @closed message of the
door is appended to the room description, [C] lines on the blackboard
are numbered, [D] classroom owner and authorized individuals will be
hushed during session like the rest of the class.";
if (callers() && callers()[1][1] != this)
  return E_PERM;
elseif (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (verb[2] == "i")
  p = "hide_when_in_session";
```

```
  blah = {"Classes in session are ", " hidden from @who listings."};
elseif (verb[2] == "s")
  p = "show_door";
  blah = {"The door is ", " displayed with the room description."};
elseif (verb[2] == "n")
  p = "numbered_bb";
  blah = {"Lines on the blackboard are ", " numbered."};
else
  p = "hush_controllers";
  blah = {"Room controllers are ", " hushed during session."};
endif
if (!argstr || !args)
  if (verb[2] == "i" && this.session)
    this.hide_occupants = this.hide_when_in_session;
  endif
  player:tell(blah[1], "currently", tostr(this.(p) ? "" | " not"),
blah[2]);
elseif (!(choice = argstr in {"off", "on"}))
  player:tell("Usage:  '", verb, " on' or '", verb, " off'");
else
  this.(p) = choice - 1;
  if (verb[2] == "i" && this.session)
    this.hide_occupants = choice - 1;
  endif
  player:tell(blah[1], tostr(this.(p) ? "now" | "no longer"), blah[2]);
endif
"Last modified Thu Jan 18 17:57:21 1996 EST by Ulf (#238@DU_MainMOO).";
.

@verb #211:"@addspeaker-s*eats @rmspeaker-s*eats @speaker-s*eats" any
any any "rd"
@prog #211:@addspeaker-seats
"Usage:  @addspeaker-seats <seat(s)>";
"        @rmspeaker-seats <seat(s)>";
"        @speaker-seats";
"Add or remove <seat(s)> as \"speaker-seats\".  Such \"speaker-seats\"
allow individuals to talk freely without being stifled whether they are
authorized users or not.";
"@speaker-seats lists the currently designated speaker-seats of the
room.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  if (verb[2] == "s")
    if (!this.speaker_seats)
      return player:tell("There are currently no speaker-seats in this
room.  Add seats with \"@addspeaker-seats <seat(s)>\"");
    else
      seats = {};
      for seat in (this.speaker_seats)
        seats = {@seats, this.details[seat][1]};
      endfor
      return player:tell("Speaker-seats for ", $string_utils:nn(this),
": ", $string_utils:english_list(seats), ".");
```

```
    endif
  else
    player:tell("Usage:  @addspeaker-seats <seat(s)>");
    player:tell("        @rmspeaker-seats <seat(s)>");
    return player:tell("        @speaker-seats");
  endif
else
  seats = $string_utils:words($string_utils:subst(argstr, {{",", "
"}}));
  seats = typeof(seats) == STR ? {seats} | seats;
  if (verb[2] == "r")
    removed = {};
    notss = {};
    notfound = {};
    for seat in (seats)
      if (s = this:match_detail(seat))
        if (s in this.speaker_seats)
          this.speaker_seats = setremove(this.speaker_seats, s);
          removed = setadd(removed, this.details[s][1]);
        else
          notss = setadd(notss, this.details[s][1]);
        endif
      else
        notfound = setadd(notfound, tostr("\"", seat, "\""));
      endif
    endfor
    if (notfound)
      player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 2 ? " don't seem to be details." | " doesn't
seem to be a detail."));
    endif
    if (notss)
      player:tell($string_utils:english_list(notss),
tostr(length(notss) > 2 ? " don't seem to be speaker-seats." | "
doesn't seem to be a speaker-seat."));
    endif
    if (removed)
      player:tell("You remove ", $string_utils:english_list(removed), "
as speaker-seats.");
    endif
  else
    added = {};
    added_already = {};
    notfound = {};
    for seat in (seats)
      if (s = this:match_detail(seat))
        if (!(s in this.speaker_seats))
          this.speaker_seats = setadd(this.speaker_seats, s);
          added = setadd(added, this.details[s][1]);
        else
          added_already = setadd(added_already, tostr("\"",
this.details[s][1], "\""));
        endif
      else
        notfound = setadd(notfound, tostr("\"", seat, "\""));
```

```
      endif
    endfor
    if (notfound)
      player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 2 ? " don't seem to be details." | " doesn't
seem to be a detail."));
    endif
    if (added_already)
      player:tell($string_utils:english_list(added_already),
tostr(length(added_already) > 2 ? " are already designated speaker-
seats." | " is alread a designated speaker-seat."), "  Type @speaker-
seats to get a list of designated speaker-seats.");
    endif
    if (added)
      player:tell("You add ", $string_utils:english_list(added), " to
the list of speaker-seats.");
    endif
  endif
endif
"Last modified Thu Oct 12 12:52:46 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"@restrict*ions @unrestrict" any any any "rd"
@prog #211:@restrictions
"Usage:  @restrict <verb(s)>";
"        @unrestrict <verb(s)>";
"        @restrictions";
"Restrict or unrestrict the usage of the specified <verb(s)> to
authorized users of the classroom.";
"If \"sit\" is restricted, only authorized users may sit at speaker-
seats, such as the \"teacher's desk\".";
"If \"open\" is restricted, only authorized users may open and close
the door to the classroom, and hence manipulate the classroom
security.";
"@restrictions will display the list of currently restricted verbs.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  if (verb = "@restrictions")
    verbs = this:list_all_verb_names(this.restricted_verbs);
    names = {};
    for v in (verbs)
      names = {@names, (len = length(v)) > 1 ? tostr(v[1], " (aka ",
$string_utils:english_list(v[2..len]), ")") | v[1]};
    endfor
    player:tell("The following verbs are restricted to authorized
users: ", $string_utils:english_list(names, "none"), ".");
  else
    player:tell("Usage:  @restrict <verb(s)>");
    player:tell("        @unrestrict <verb(s)>");
    return player:tell("        @restrictions");
  endif
else
```

```
  actions = $string_utils:words($string_utils:subst(argstr, {{",", "
"}}));
  actions = this:verb_names(actions);
  actions = typeof(actions) == STR ? {actions} | actions;
  if (verb[2] == "u")
    removed = {};
    notfound = {};
    for action in (actions)
      if (action in this.restricted_verbs)
        this.restricted_verbs = setremove(this.restricted_verbs,
action);
        removed = setadd(removed, action);
      else
        notfound = setadd(notfound, tostr("\"", action, "\""));
      endif
    endfor
    if (notfound)
      player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 2 ? " are not currently restricted verbs." | "
is not a currently restricted verb."), "  Type @restrictions to get a
list of currently restricted verbs.");
    endif
    if (removed)
      player:tell("You remove the restrictions for ",
$string_utils:english_list(removed), ".");
    endif
  else
    added = {};
    added_already = {};
    for action in (actions)
      if (!(action in this.restricted_verbs))
        this.restricted_verbs = setadd(this.restricted_verbs, action);
        added = setadd(added, action);
      else
        added_already = setadd(added_already, tostr("\"", action,
"\""));
      endif
    endfor
    if (added_already)
      player:tell($string_utils:english_list(added_already),
tostr(length(added_already) > 2 ? " are already restricted verbs." | "
is alread a restricted verb."), "  Type @restrictions to get a list of
currently restricted verbs.");
    endif
    if (added)
      player:tell("You restrict ", $string_utils:english_list(added),
".");
    endif
  endif
endif
"Last modified Thu Oct 12 12:51:04 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"@stifle @unstifle*d" any any any "rd"
```

```
@prog #211:@stifle
"Usage:  @stifle <verb(s)>";
"        @unstifle <verb(s)>";
"        @unstifled";
"Stifle or unstifle the output of <verb(s)> for a classroom in
session.";
"Unstifling the output of verbs such as \"@go\" or \"home\" is
recommended, if you want to avoid silent teleportation into and out of
the classroom.";
"@unstifled will display the currently unstifled verbs for the
classroom.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  if (verb = "@unstifled")
    verbs = this:list_all_verb_names(this.allowed_sources);
    names = {};
    for v in (verbs)
      names = {@names, (len = length(v)) > 1 ? tostr(v[1], " (aka ",
$string_utils:english_list(v[2..len]), ")") | v[1]};
    endfor
    player:tell($string_utils:nn(this), " allows output from the
following verbs while in session:  ", $string_utils:english_list(names,
"none"), ".");
    player:tell("Please consult `help moderation' for more information
on output stifling.");
  else
    player:tell("Usage:  @stifle <verb(s)>");
    player:tell("        @unstifle <verb(s)>");
    player:tell("        @unstifled");
  endif
else
  sources = $string_utils:words($string_utils:subst(argstr, {{",", "
"}}));
  sources = this:verb_names(sources);
  sources = typeof(sources) == STR ? {sources} | sources;
  if (verb[2] == "s")
    removed = {};
    notfound = {};
    for source in (sources)
      if (match = $list_utils:assoc_prefix(source,
this.data.verb_groups))
        for v in (match[2])
          if (v in this.allowed_sources)
            this.allowed_sources = setremove(this.allowed_sources, v);
            removed = setadd(removed, v);
          else
            notfound = setadd(notfound, v);
          endif
        endfor
      elseif (source in this.allowed_sources)
        this.allowed_sources = setremove(this.allowed_sources, source);
        removed = setadd(removed, source);
      else
```

```
        notfound = setadd(notfound, source);
      endif
    endfor
    if (notfound)
      player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 2 ? " aren't unstifled verbs." | " isn't a
unstifled verb."), "  Type @unstifled to get a list of currently
unstifled verbs.");
    endif
    if (removed)
      player:tell("You stifle the output of ",
$string_utils:english_list(removed), ".");
    endif
  else
    added = {};
    added_already = {};
    for source in (sources)
      if (match = $list_utils:assoc_prefix(source,
this.data.verb_groups))
        for v in (match[2])
          if (!(v in this.allowed_sources))
            this.allowed_sources = setadd(this.allowed_sources, v);
            added = setadd(added, v);
          else
            added_already = setadd(added_already, v);
          endif
        endfor
      elseif (!(source in this.allowed_sources))
        this.allowed_sources = setadd(this.allowed_sources, source);
        added = setadd(added, source);
      else
        added_already = setadd(added_already, source);
      endif
    endfor
    if (added_already)
      player:tell($string_utils:english_list(added_already),
tostr(length(added_already) > 2 ? " are already unstifled verbs." | "
is alread an unstifled source."), "  Type @unstifled to get a list of
currently unstifled verbs.");
    endif
    if (added)
      player:tell("You unstifle the output of ",
$string_utils:english_list(added), ".");
    endif
  endif
endif
"Last modified Fri Oct 13 16:23:50 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"@auth*orized @unauth*orize" any any any "rd"
@prog #211:@authorized
"Usage:  @authorize <user(s)>";
"        @unauthorize <user(s)>";
"        @authorized";
```

```
"Authorize or unauthorize <user(s)> for the classroom.";
"Authorized users may use all the administrative verbs.";
"@authorized displays the list of currently authorized users.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  if (verb == "@authorized")
    group = this.authorized;
    player:tell("Authorized users of ", $string_utils:nn(this), ": ",
tostr(group ? $string_utils:names_of(group, ", ") | "nobody"), ".");
  else
    player:tell("Usage:  @authorize <user(s)>");
    player:tell("        @unauthorize <user(s)>");
    return player:tell("        @authorized");
  endif
else
  group = $string_utils:words($string_utils:subst(argstr, {{",", "
"}}));
  notfound = {};
  rest = {};
  if (verb[2] == "u")
    removed = {};
    for user in (group)
      if (valid(person = player:my_match_object(user)) || valid(person
= $player_db:find(user)))
        if (person in this.authorized)
          this.authorized = setremove(this.authorized, person);
          removed = setadd(removed, person);
        else
          rest = setadd(rest, person);
        endif
      else
        notfound = setadd(notfound, tostr("\"", user, "\""));
      endif
    endfor
    if (notfound)
      player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 1 ? " are not the names of any users." | " is
not the name of any user."));
    endif
    if (rest)
      player:tell($string_utils:names_of(rest, ", "),
tostr(length(rest) > 2 ? " are not currently authorized users." | " is
not a currently authorized user."), "  Type @authorized to get a list
of currently authorized users.");
    endif
    if (removed)
      player:tell("You remove ", $string_utils:names_of(removed, ", "),
" from the list of authorized users.");
    endif
  else
    added = {};
    for user in (group)
```

```
      if (valid(person = player:my_match_object(user)) || valid(person
= $player_db:find(user)))
        if (!(person in this.authorized))
          this.authorized = setadd(this.authorized, person);
          added = setadd(added, person);
        else
          rest = setadd(rest, person);
        endif
      else
        notfound = setadd(notfound, tostr("\"", user, "\""));
      endif
    endfor
    if (notfound)
      player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 1 ? " are not the names of any users." | " is
not the name of any user."));
    endif
    if (rest)
      player:tell($string_utils:names_of(rest, ", "),
tostr(length(rest) > 2 ? " are already authorized users." | " is
already an authorized user."), "  Type @authorized to get a list of
currently authorized users.");
    endif
    if (added)
      player:tell("You add ", $string_utils:names_of(added, ", "), " to
the list of authorized users.");
    endif
  endif
endif
"Last modified Thu Aug  3 23:59:59 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"@session" any any any "rd"
@prog #211:@session
"Usage:  @session begin|end";
"        @session";
"Begin or end a class-session.  Only unstifled verbs' output is seen
from students (\"emote\" and \"say\" are restricted to tables students
sit at); authorized users' output is not stifled.";
"This may also toggle the visibility of people in the room in @who
listings, if the room is set up to hide its occupants in sessions.  The
room title changes to display the @session message behind the room
name.";
"@session all by itself displays the current status of the classroom.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  return player:tell($string_utils:nn(this), " is currently ",
this.session ? "in" | "not in", " session.");
endif
if (dobj == this && prepstr == "is")
  return $last_huh:(verb)(@args);
elseif (argstr in {"on", "start", "begin", "1"})
  if (this.session)
```

```
    player:tell(this:title(), " is already in session.");
  else
    this.session = 1;
    this.hide_occupants = this.hide_when_in_session ? 1 |
this.hide_occupants;
    this:announce_all(this:session_begin_msg());
  endif
elseif (argstr in {"off", "stop", "end", "0"})
  if (!this.session)
    player:tell(this:title(), " is not in session currently.");
  else
    this.session = 0;
    this.hide_occupants = this.hide_when_in_session ? 0 |
this.hide_occupants;
    this:announce_all(this:session_end_msg());
  endif
else
  player:tell("Usage:  ", verb, " begin|end");
endif
"Last modified Fri Aug  4 00:00:01 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"@mkclass" any any any "rd"
@prog #211:@mkclass
"Usage:  @mkclass <class-name>";
"        @mkclass <VSPO-group-number>";
"Set up a new class for the classroom.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  return player:tell("Usage:  ", verb, " <class-name>|<VSPO-group-
number>");
endif
class = this:find_class(argstr);
if (typeof(class) == NUM)
  if ((name = this.classes[class][1]) == argstr)
    return player:tell("Sorry, there is already a class by that
name.");
  elseif (!$command_utils:yes_or_no(tostr("There's already a class with
a similar name - \"", name, "\".  Do you want to add a new class called
\"", argstr, "\" anyway?")))
    return player:tell("Aborted.");
  endif
elseif (class)
  matches = {};
  for i in (class)
    matches = {@matches, this.classes[i][1]};
  endfor
  if (!$command_utils:yes_or_no(tostr("There are already classes with
similar names: ", $string_utils:english_list(matches), ".  Do you want
to add a new class called \"", argstr, "\" anyway?")))
    return player:tell("Aborted.");
  endif
```

```
else
  if ($string_utils:is_numeric(argstr) &&
$command_utils:yes_or_no("Would you like to set up a class for a VSPO
group?"))
    if ((group = $vspo_registry:get_group(gnum = tonum(dobjstr))) ==
E_INVARG)
      return player:tell("There isn't a VSPO group numbered \"",
argstr, "\".");
    endif
  else
    group = {};
  endif
  if (group)
    this.classes = setadd(this.classes, {@group[2..3], {time(),
time()}, player});
    this.current_class = this:find_class(group[2]);
  else
    this.classes = setadd(this.classes, {argstr, {}, {time(), time()},
player});
    this.current_class = this:find_class(argstr);
  endif
  newclass = group ? group[2] | argstr;
  player:tell("You set up a new class called ", newclass, ".");
  this:announce(player.name, " sets up a new class called ", newclass,
".");
endif
"Last modified Fri Aug  4 00:00:03 1995 EDT by Ulf (#238@DU_Moo-
East).";
.
@verb #211:"@rmclass" any any any "rd"
@prog #211:@rmclass
"Usage:  @rmclass <class-name>";
"Remove a registered class from the classroom.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  return player:tell("Usage:  ", verb, " <class-name>");
endif
class = this:find_class(argstr);
if (!class)
  player:tell("Sorry, there does not appear to be a class by that
name.");
elseif (typeof(class) == NUM)
  this.classes = listdelete(this.classes, class);
  this.current_class = this.classes ? 1 | 0;
  player:tell("You remove a class called ", argstr, ".");
  this:announce(player.name, " removes a class called ", argstr, ".");
else
  matches = {};
  for i in (class)
    matches = {@matches, this.classes[i][1]};
  endfor
  player:tell("Please be more specific.");
```

```
  player:tell("\"", argstr, "\" could refer to ",
$string_utils:english_list(matches, "none", " or "), ".");
endif
"Last modified Fri Aug  4 00:00:10 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"@class*es @setup-c*lass @setupc*lass" any any any "rd"
@prog #211:@classes
"Usage:  @class <class-name>";
"        @class";
"        @classes";
"Set up a classroom for <class-name>.  If the classroom is in session,
only students of <class-name> will be allowed to enter the classroom;
authorized users may always enter, regardless.";
"@class all by itself displays the class the classroom is currently set
up for.  @classes displays a list of registered classes complete with
the number of students associated with each of them.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  if (!this.classes)
    return player:tell("Sorry, there haven't been any classes set up
yet.");
  elseif (verb == "@class")
    if (this.current_class)
      return player:tell($string_utils:nn(this), " is currently set up
for ", this.classes[this.current_class][1], ".");
    else
      return player:tell($string_utils:nn(this), " is not set up for
any class, currently.");
    endif
  else
    classes = {};
    for class in (this.classes)
      classes = {@classes, tostr(class[1], " [", tostr(length(class[2])
? length(class[2]) | "no"), " students]")};
    endfor
    return player:tell("Registered classes for ",
$string_utils:nn(this), ": ", $string_utils:english_list(classes),
".");
  endif
endif
class = this:find_class(argstr);
if (typeof(class) == NUM)
  this.current_class = class;
  player:tell("You set ", this:title(), " up for ",
this.classes[class][1], ".");
  this:announce(player.name, " sets ", this:title(), " up for ",
this.classes[class][1], ".");
elseif (!class)
  player:tell("Sorry, there does not appear to be any class by that
name.");
else
  matches = {};
```

```
  for i in (class)
    matches = {@matches, this.classes[i][1]};
  endfor
  player:tell("Please be more specific.");
  player:tell("\"", argstr, "\" could refer to ",
$string_utils:english_list(matches, "none", " or "), ".");
endif
"Last modified Sun Jan 21 23:54:38 1996 EST by Ulf (#238@DU_MainMOO).";
.

@verb #211:"reg*ister unreg*ister" any none none "rd"
@prog #211:register
"Usage:  register <user(s)>";
"        unregister <user(s)>";
"Add or remove <user(s)> from the roster of the currently set up
class.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
if (!argstr)
  return player:tell("Usage:  ", verb, " <student(s)>");
else
  group = $string_utils:words($string_utils:subst(argstr, {{",", "
"}}));
  notfound = {};
  rest = {};
  class = this.current_class;
  if (class && this.classes[class])
    if (verb[1] == "u")
      removed = {};
      for user in (group)
        if (valid(person = $string_utils:match(user,
this.classes[class][2], "name", this.classes[class][2], "aliases")) ||
valid(person = player:my_match_object(user)) || valid(person =
$player_db:find(user)))
          if (person in this.classes[class][2])
            this.classes[class][2] = setremove(this.classes[class][2],
person);
            removed = setadd(removed, person);
          else
            rest = setadd(rest, person);
          endif
        else
          notfound = setadd(notfound, tostr("\"", user, "\""));
        endif
      endfor
      if (notfound)
        player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 1 ? " are not the names of any users." | " is
not the name of any user."));
      endif
      if (rest)
        player:tell($string_utils:names_of(rest, ", "),
tostr(length(rest) > 2 ? " are not registered to " | " is not
registered to "), this.classes[class][1], ".");
```

```
      endif
      if (removed)
        this.classes[class][3][2] = time();
        player:tell("You remove ", $string_utils:names_of(removed, ",
"), " from ", this.classes[class][1], ".");
      endif
    else
      added = {};
      for user in (group)
        if (valid(person = player:my_match_object(user)) ||
valid(person = $player_db:find(user)))
          if (!(person in this.classes[class][2]))
            this.classes[class][2] = setadd(this.classes[class][2],
person);
            added = setadd(added, person);
          else
            rest = setadd(rest, person);
          endif
        else
          notfound = setadd(notfound, tostr("\"", user, "\""));
        endif
      endfor
      if (notfound)
        player:tell($string_utils:english_list(notfound),
tostr(length(notfound) > 1 ? " are not the names of any users." | " is
not the name of any user."));
      endif
      if (rest)
        player:tell($string_utils:names_of(rest, ", "),
tostr(length(rest) > 2 ? " are already registered for " | " is already
registered for "), this.classes[class][1], ".");
      endif
      if (added)
        this.classes[class][3][2] = time();
        player:tell("You add ", $string_utils:names_of(added, ", "), "
to ", this.classes[class][1], ".");
      endif
    endif
  else
    if (this.classes)
      player:tell("Please set up a class with @class <classname>
first.");
    else
      player:tell("Please set up a class with @mkclass <classname>
first.");
    endif
  endif
endif
"Last modified Sat Feb  3 12:19:58 1996 EST by Ulf (#238@DU_MainMOO).";
.

@verb #211:"@stat*us" any any any "rd"
@prog #211:@status
"Usage:  @status [long|verbose]";
```

```
"Display the classroom's setup.  If \"long\" is given as an argument
you get the extended version.";
if (!this:is_authorized(player, verb))
  return player:tell(this:restricted_msg());
endif
su = $string_utils;
long = index(argstr, "long") || index(argstr, "verbose") ? 1 | 0;
player:tell("Status for ", su:nn(this), ":");
player:tell();
player:tell("The room is ", tostr(this.session ? "" | "not "), "in
session and the door is presently ", tostr(this.door_open ? "open." |
"closed."));
player:tell();
if (long)
  this:("@invisible")();
  player:tell();
  this:("@show-d")();
  player:tell();
  this:("@numbers")();
  player:tell();
  this:("@hush-c")();
  player:tell();
endif
if (len = length(this.allowed_sources))
  if (!long)
    player:tell(len, " verbs may send text to the room in session.");
    player:tell("  To see a list of them, type: @unstifled");
  else
    verbs = this:list_all_verb_names(this.allowed_sources);
    names = {};
    for v in (verbs)
      names = {@names, (len = length(v)) > 1 ? tostr(v[1], " (aka ",
su:english_list(v[2..len]), ")") | v[1]};
    endfor
    player:tell("It allows output from the following verbs while in
session:  ", su:english_list(names, "none"), ".");
  endif
else
  player:tell("No verbs may send text to the room in session.");
endif
player:tell();
if (len = length(this.restricted_verbs))
  if (!long)
    player:tell("The use of ", len, " verbs is currently restricted.");
    player:tell("  To see a list of them, type: @restrictions");
  else
    verbs = this:list_all_verb_names(this.restricted_verbs);
    names = {};
    for v in (verbs)
      names = {@names, (len = length(v)) > 1 ? tostr(v[1], " (aka ",
su:english_list(v[2..len]), ")") | v[1]};
    endfor
    player:tell("The following verbs are restricted to authorized
users: ", su:english_list(names, "none"), ".");
  endif
```

```
else
  player:tell("None of the room's verbs are restricted.");
endif
player:tell();
if (this.current_class)
  player:tell("It can be setup for any of the following classes:");
  player:tell();
  for c in [1..length(this.classes)]
    player:tell("    ", su:left(this.classes[c][1], 25),
length(this.classes[c][2]), " registered");
  endfor
  player:tell();
  player:tell("It is currently setup for ",
this.classes[this.current_class][1], ".");
  player:tell();
  if (bunch = this.classes[this.current_class][2])
    nameless = 0;
    for student in (bunch)
      if (!student.name)
        bunch = setremove(bunch, student);
        nameless = nameless + 1;
      endif
    endfor
    player:tell("Class roster includes:  ", su:title_list(bunch),
tostr(nameless ? tostr(" [+", nameless, " uninitialized VSPOs]") |
""));
  else
    player:tell("Roster is empty.");
  endif
elseif (!this.classes)
  player:tell("No classes have been set up here yet.");
else
  player:tell("The classroom is not set up for any of the ",
length(this.classes), " classes registered.");
endif
player:tell();
if (this.authorized)
  player:tell("Authorized users: ", su:title_list(this.authorized));
else
  player:tell("The list of authorized users is empty.");
endif
player:tell();
"Last modified Mon Jan 22 00:13:00 1996 EST by Ulf (#238@DU_MainMOO).";
.

@verb #211:"trusted" this none this "rx"
@prog #211:trusted
cv = callers()[1][2];
return $perm_utils:controls(args[1], this) || args[1] in this.residents
|| this:is_authorized(args[1], cv);
"Last modified Fri Oct 27 13:18:21 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #211:"init_for_core" this none this "rx"
```

```
@prog #211:init_for_core
"Copied from The System Object (#0):init_for_core by Administrator (#2)
Mon Dec  1 16:26:51 1997 EST";
if (caller_perms().wizard)
  pass();
  if ("server_started" in verbs(this))
    code = {"callers() || ($last_restart_time = time());"};
    set_verb_code(this, "server_started", code);
  endif
  if ("name_lookup_failed" in verbs(this))
    delete_verb(this, "name_lookup_failed");
  endif
  if ("uptime_since" in verbs(this))
    delete_verb(this, "uptime_since");
  endif
  if ("do_command" in verbs(this))
    delete_verb(this, "do_command");
  endif
  $shutdown_message = "";
  $shutdown_time = 0;
  $dump_interval = 3600;
  $gripe_recipients = {player};
  $class_registry = {{"generics", "Generic objects intended for use as
the parents of new objects", {$room, $exit, $thing, $note, $letter,
$container, $root_class, $player, $prog, $wiz, $generic_editor,
$mail_recipient, $mail_agent}}, {"utilities", "Objects holding useful
general-purpose verbs", {$string_utils, $gender_utils, $trig_utils,
$time_utils, $match_utils, $object_utils, $lock_utils, $list_utils,
$command_utils, $code_utils, $perm_utils, $building_utils}}};
  set_verb_code(this, "user_connected", verb_code(this,
"user_connected(core)"));
  delete_verb(this, "user_connected(core)");
endif
.

@verb #211:"@rmdetail" any any any "rd"
@prog #211:@rmdetail
pass(@args);
"Last modified Thu Mar 26 09:13:44 1998 EST by Creeper
(#101@Virtual_Campus).";
.

----- End of @archive #211
```

$storage

```
@archive #185 with verbs and properties
----- Beginning of @archive #185

@create #3 named Storage Area Prototype,storage
;#185.("entrances") = {#1054}
;#185.("blessed_object") = #-1
;#185.("blessed_task") = 1744126431
;#185.("exits") = {#790}
;#185.("invited") = {}
```

```
;#185.("key") = 0
;#185.("aliases") = {"Storage Area Prototype", "storage"}
;#185.("description") = {"This area, a storage area, allows you to
register objects according to certain categories, and have them
displayed when needed.", "To see the displays available type:
displays", "If you need additional help type:   help here"}
;#185.("object_size") = {30982, 896601616}
;#185.("web_calls") = 155
;#185.("last_modified") = 880691499
;#185.("creation_date") = 0
@prop #185."open" 0 "rc"
@prop #185."displays" {{}} "r"
@prop #185."not_add_msg" "You can't seem to add that here." "rc"
@prop #185."not_control_msg" "You are not allowed to do that." "rc"
@prop #185."single" 0 "rc"
@prop #185."names" {} "r"
@prop #185."help_single_msg" {"This is a display room in which you can
obtain information about various objects that are put up on display by
their respective creators.", "", "To see what objects are registered
here, type 'displays'.", "To have a closer look at one of the objects
type 'show <object>' where <object> is either its number in the display
list, or some part of its name."} "rc"
@prop #185."help_multi_msg" {"This is a display room that holds lists
of different objects.  The objects are separated into different
categories.  To see a list of categories,  type: 'displays'.  To see a
list of objects on a particular category, type 'display <category>',
where <category> is either the number of that category in the list, or
a distinguishable part of its name.  To see a listing of all
categories, type 'display all'.", "", "To see more information on a
particular object, type: 'show <object>', where <object> is either that
object's number or part of its name."} "rc"
@prop #185."help_control_msg" {"", "The display room has a toggle
between having only one list of objects, or allowing more lists.  This
toggle is controlled by the property .single.  Set the .single property
to 0 if you would like more lists of objects in the displays.", "", "If
you choose to do that, additional commands that control categories
become available;", "", "'@add-cat*egory <category name>'", "
adds <category name> as the name of a new category.", "'@rm-cat*egory
<category name>'", "        removes the category by the name <category
name>.", "'@rename-cat*egory <category name> to <new name>'", "
changes the name of that category.", "", "And a very powerful verb for
easy resetting of the displays, '@clear'.This will clear off all
displays.  It will ask for confirmation, but be careful.", "", "To
decide which players and objects that can be added to the displays,
there are two verbs that controls this.  :check_player, will return a
true value if the person doing a registration is allowed to add the
object *e is trying to add.  And :check_object will return 1 if the
object can be added.  As for now, there is only one way to protect
registration without programming one of the controls verb.  Set the
.open property to 0, and only someone who controls the room can
register objects.  The person trying to register an object will then
see the .register_msg.", "", "If other tests on objects or persons are
wanted, like only fertile objects, talk to Creeper, to make a
restricted kid of this generic with the wanted tests.  Or you can
program the :check_* verbs yourself."} "rc"
@prop #185."register_msg" {"", "To register an object here, talk to the
owner of this place."} "rc"
```

```
@prop #185."help_register_msg" {"", "To register an object here, type
'@register <object>', where <object> is the", "object number, or name,
of the object you would like to register.", "", "To remove an object
you have registered, type '@remove <object>', where <object>", "is
either its number in the display list, or a prefix of its name."} "rc"
@prop #185."register_multi_help_msg" {"", "To register an object here,
type 'register <object> for <category>', where ", "<object> is the
object you would like to add, and <category> is the category to", "put
it in.  You'll see the list of categories by typing 'displays'.", "",
"To remove an object you have added, type 'remove <object> from
<category>'.", "where <object> and <category> are as in register."}
"rc"
@prop #185."help_check_msg" {"", "For an object to be added to this
display room, it has to be of a special kind.  The object has to be
readable, has to have at least one verb defined on it, has to have help
documentation, be a child of here.generic_parent, and either be a
feature object or fertile."} "rc"
@prop #185."public_library" 0 "rc"
@prop #185."current_category" {} "r"
@prop #185."generic_parent" #1 "rc"
@prop #185."help_required" 1 "rc"

@verb #185:"check_object" this none this "rxd"
@prog #185:check_object
"Adds extra tests to the permisions check from a level down";
"Objects have to be readable, fertile or feature object, have to have
help (depends on .help_required), have to have at least one verb
defined, and be a child of .generic_parent.";
"this.generic_parent can either be an object number (the required
parent) or a list of object numbers (all acceptable parents).";
object = args[2];
if (typeof(this.generic_parent) != LIST && !$object_utils:isa(object,
this.generic_parent))
  player:tell("Objects that are to be added here have to be descendents
of ", $string_utils:nn(this.generic_parent), ".");
  return 0;
elseif ($object_utils:isa(object, $feature))
  if (!(object.feature_ok && object.r))
    player:tell("Features have to have their .feature_ok property set
and have to be readable.");
    return 0;
  endif
elseif (!(object.f && object.r))
  player:tell("Objects have to be fertile and readable.");
  return 0;
endif
if (length(verbs(object)) < 1)
  player:tell("Objects have to have at least one verb added.");
  return 0;
elseif (this.help_required &&
typeof($code_utils:verb_or_property(object, "help_msg")) == ERR)
  if (!(player.wizard && $command_utils:yes_or_no("The object seems to
lack help documentation. Add it anyway?")))
    player:tell("Objects have to contain sufficient help
documentation.");
    return 0;
  endif
```

```
endif
if (this.generic_parent && typeof(this.generic_parent) == LIST)
  for parent in (this.generic_parent)
    if ($object_utils:isa(object, parent))
      return 1;
    endif
  endfor
  player:tell("Object must be decended from one of one of these
objects: ", $string_utils:nn_list(this.generic_parent, "<error>", " or
"));
  return 0;
endif
return 1;
"Last modified Tue May 20 13:14:03 1997 EDT by EricM
(#5058@DU_MainMOO).";
.

@verb #185:"check_player" this none this "rxd"
@prog #185:check_player
if ((c = callers()) && c[1][2] == "help_msg")
  return 1;
else
  return this.open || $perm_utils:controls(args[1], this) ||
(length(args) > 1 ? $perm_utils:controls(args[1], args[2]) | 0);
endif
"Last modified Fri Nov  3 10:34:32 1995 EST by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"display*s" any any any "rd"
@prog #185:displays
"Usage:  displays [<category name>]";
"Gives a listing of the displays in this room.  <category name> is used
when the room has more than one category.";
if (!this.single)
  if (!argstr)
    "Wants a list of the displays.";
    out = {};
    for i in [1..len = length(this.names)]
      out = {@out, tostr(" ", i, ")",
$string_utils:space(length(tostr(len)) + 2 - (length(tostr(i)) + 1), "
"), this.names[i])};
    endfor
    player:tell("The following categories are available:");
    player:tell();
    player:tell_lines($string_utils:columnize(out, 2,
player:linelen()));
    player:tell();
    player:tell("To see the displays on one of them, type 'display
<category>'");
    return;
  elseif (argstr == "all")
    "Wants a run-down of all displays.";
    for i in [1..length(this.names)]
      cur = this:get_category();
```

```
      player:tell_lines({tostr(tostr(i == cur ? "*" | ""), i, ")
Displays on ", this.names[i], ":"), "",
@this:preper_show(this.displays[i], 0),
$string_utils:space(player:linelen(), "-"), ""});
    endfor
    return player:tell("Done showing all available categories;  \"*\"
marks the category you are currently viewing.  Type 'display
<category>' to view categories individually.");
  elseif (!(ind = this:find_cat(argstr)))
    what = $string_utils:is_numeric(argstr) ?
$string_utils:english_ordinal(argstr) | tostr("\"", argstr, "\"");
    player:tell("There is no ", what, " category to display.");
    player:tell("Type 'displays' for a listing of available
categories.");
    if ("on" in args)
      player:tell("Maybe you mean to 'show ", argstr, "'?");
    endif
    return;
  else
    player:tell_lines({tostr("Displays on ", this.names[ind], ":"), "",
@this:preper_show(this.displays[ind], 0),
$string_utils:space(player:linelen(), "-"), "To look closer at one of
these objects, type 'show <object>'."});
    this:set_category(ind);
  endif
elseif (argstr)
  player:tell("No arguments to display, maybe you mean to 'show ",
argstr, "'?");
else
  player:tell_lines({@this:preper_show(this.displays[1], 0),
$string_utils:space(player:linelen(), "-"), "To look closer at one of
the objects, type 'show <object>'."});
endif
"Last modified Thu Jun 29 15:07:51 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"@add-cat*egory" any any any "rd"
@prog #185:@add-category
"Usage:  @add-cat*egory <category name>";
"Makes a new category for adding displays.";
if (!$perm_utils:controls(player, this))
  return player:tell(E_PERM);
elseif (this.single)
  return player:tell("The room is in single modus.");
elseif (argstr in this.names)
  return player:tell(argstr, " is already a category.");
else
  this.names = {@this.names, argstr};
  this.displays = {@this.displays, {}};
  player:tell("Category \"", argstr, "\" added.");
endif
"Last modified Wed Jun 28 12:31:39 1995 EDT by Ulf (#238@DU_Moo-
East).";
.
```

```
@verb #185:"@rm-cat*egory" any any any "rd"
@prog #185:@rm-category
"'@rm-cat*egory <category name>'  Removes that category.";
if (!$perm_utils:controls(player, this))
  return player:tell(E_PERM);
elseif (this.single)
  return player:tell("The room is in single modus.");
elseif (!(cat = this:find_cat(argstr)))
  return player:tell(argstr, " is not a category.");
else
  this.names = listdelete(this.names, cat);
  this.displays = listdelete(this.displays, cat);
  if (length(this.displays) == 0)
    this.displays = {{}};
  endif
  player:tell("Category removed.");
endif
"Last modified Tue Jun 27 13:09:01 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"show" any any any "rd"
@prog #185:show
"Usage:  show <object> [on category]";
"Checks for .generic_parent to decide whether to display descendents
number or not.";
if (!dobjstr)
  return player:tell("You need to specify an object to look at.");
elseif (!this.single)
  if (!(ind = $string_utils:index_delimited(argstr, "on")) && !(cur =
this:get_category()))
    return player:tell("Usage:  show <object> on <category>");
  elseif (!(cat = this:find_cat(catstr = argstr[ind +
3..length(argstr)])) && ind)
    what = $string_utils:is_numeric(catstr) ?
$string_utils:english_ordinal(catstr) | tostr("\"", catstr, "\"");
    player:tell("There is no ", what, " category to look at.");
    return player:tell("Type 'displays' for a listing of available
categories.");
  else
    object = this:match_disp(dobjstr, cat ? cat | cur);
    cat = cat ? cat | cur;
  endif
else
  object = this:match_disp(dobjstr);
  cat = 1;
endif
linelen = player:linelen();
if (typeof(object) != OBJ)
  if (object)
    out = {};
    for o in (object)
      out = {@out, tostr($string_utils:left(tostr(" ", o in
this.displays[cat], ")"), 4), " ", o:title())};
    endfor
```

```
    player:tell("I don't know which of the following you were referring
to:");
    player:tell_lines($string_utils:columnize(out, 2, linelen));
    return;
  else
    return player:tell("I don't know what object you are referring to
with \"", dobjstr, "\".");
  endif
elseif (object == $failed_match)
  if (!this.single)
    if ($string_utils:is_numeric(dobjstr))
      player:tell("There is no ",
$string_utils:english_ordinal(dobjstr), " object in the ",
this.names[cat], " category.");
      player:tell("Type 'display ", this.names[cat], "' to list the
objects in that category.");
    else
      player:tell("\"", dobjstr, "\" doesn't match any object in the ",
this.names[cat], " category.");
    endif
  else
    if ($string_utils:is_numeric(dobjstr))
      player:tell("There is no ",
$string_utils:english_ordinal(dobjstr), " object in the displays.");
      player:tell("Type 'displays' to list the displayed objects.");
    else
      player:tell("\"", dobjstr, "\" doesn't match any object in the
displays.");
    endif
  endif
  return;
endif
player:tell($string_utils:space(linelen, "-"));
player:tell("Name: ", object.name, " (", object, ")");
player:tell("Owner: ", object.owner.name, " (", object.owner, ")");
player:tell("Parent: ", parent(object).name, " (", parent(object),
")");
flags = "Flags: ";
if (is_player(object))
  flags = tostr(flags, "player ", object.wizard ? "wizard " | "",
object.programmer ? "programmer " | "", object.builder ? "builder " |
"");
endif
player:tell(flags, object.r ? "readable " | "", object.f ? "fertile " |
"", object.w ? "writable" | "");
if ($object_utils:isa(object, $note))
  "....This checks to see if the object is a note....";
  player:tell($string_utils:space(linelen, "-"));
  object:look_self();
  object:read();
elseif ($object_utils:isa(object, $mail_recipient) && args[1] in
properties(this))
  "....This checks to see if the object is a mail folder....";
  "....and if a particular message is to be displayed";
  player:tell($string_utils:center("Letter", linelen, "-"));
```

```
  end = this.(args[1]) + 1;
  object:display_seq_full({this.(args[1]), end}, "");
else
  player:tell("Verbs: ", length(verbs(object)));
  if (this.generic_parent != $feature || (this.generic_parent ==
$feature && children(object)))
    player:tell("Descendants: ",
length($object_utils:descendants(object)));
  endif
  ancestor = {};
  foo = object;
  while (foo != #1)
    foo = parent(foo);
    ancestor = listappend(ancestor, foo.name);
  endwhile
  player:tell("Ancestory: ", $string_utils:english_list(ancestor));
  player:tell();
  object:look_self();
endif
prop = tostr(object);
if ($object_utils:has_property(this, prop))
  player:tell($string_utils:center("Additional Comments", linelen, "-
"));
  for lines in (this.(prop))
    player:tell(lines);
  endfor
endif
player:tell($string_utils:center("finished", linelen, "-"));
if (this.generic_parent != $feature)
  hashelp = $object_utils:has_callable_verb(object, "help_msg") &&
object:help_msg();
  hasabout = $object_utils:has_callable_verb(object, "about_text") &&
object:about_text();
  help = $object_utils:has_property(object, "help_msg") &&
object.help_msg;
  about = $object_utils:has_property(object, "about_text") &&
object.about_text;
  if (help || about || hashelp || hasabout)
    help = help || hashelp;
    about = about || hasabout;
    player:tell("For more information type ", help ? tostr("\"help ",
object, "\"") | "", help && about ? " and " | "", about ?
tostr("\"@about ", object, "\"") | "");
  endif
else
  if (object.about_text && (hasabout =
$object_utils:has_callable_verb(object, "about_text")))
    player:tell("For additional information type \"@about ", object,
"\"");
  endif
endif
"Last modified Thu Oct 17 19:49:56 1996 EDT by EricM
(#5058@DU_MainMOO).";
.
```

```
@verb #185:"@register*-object" any any any "rd"
@prog #185:@register-object
"Usage:  register <object> [for category]";
"Registers a new object in this room's displays.  <object> is the
object to register.  [for category] is required if this room has more
than one category to display.";
if ($command_utils:object_match_failed(object =
player:my_match_object(dobjstr), dobjstr))
  return;
else
  if (!this.single)
    if (iobjstr && (cat = this:find_cat(iobjstr)))
      if (!this:check_player(player, object, cat))
        player:tell(this.not_control_msg);
      elseif (!this:check_object(player, object))
        player:tell(this.not_add_msg);
      else
        this:add_object(object, cat);
        player:tell("You register ", object.name, " (", tostr(object),
").");
        if (this.public_library)
          info = tostr("Registered by ", player.name, "(", player, ")
at ", ctime(), ".");
          $mail_agent:send_message(this, $system_mail_recipient,
tostr(object.name, "(", object, ") registered."), info);
        endif
      endif
    elseif (iobjstr)
      player:tell("\"", iobjstr, "\" is an unrecognized category.");
    else
      player:tell("Usage:  register <object> for <category>");
    endif
  elseif (!this:check_player(player, object))
    player:tell(this.not_control_msg);
  elseif (!this:check_object(player, object))
    player:tell(this.not_add_msg);
  else
    this:add_object(object);
    player:tell("You register ", object.name, " (", tostr(object),
").");
    if (this.public_library)
      info = tostr("Registered by ", player.name, "(", player, ") at ",
ctime(), ".");
      $mail_agent:send_message(this, $system_mail_recipient,
tostr(object.name, "(", object, ") registered."), info);
    endif
  endif
endif
"Last modified Wed Jun 28 12:50:35 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"@remove*-object" any any any "rd"
@prog #185:@remove-object
"Usage:  @remove <object> [from category]";
```

```
"To remove an object from this rooms displays.  <object> is the object
to remove.  [fromrom category] is required, if this room has more than
one category to display in.";
if (!dobjstr)
  return player:tell("Usage: @remove <object> [from <category>]");
elseif (!valid(object = this:match_disp(dobjstr)))
  if ($command_utils:object_match_failed(object =
player:my_match_object(dobjstr), dobjstr))
    return;
  endif
endif
if (!this.single)
  if (iobjstr && (cat = this:find_cat(iobjstr)))
    if (!this:check_player(player, object, cat))
      player:tell(this.not_control_msg);
    else
      this:rm_object(object, cat);
      player:tell("You remove ", object.name, " (", tostr(object),
").");
      if (this.public_library)
        info = tostr("Removed by ", player.name, "(", player, ") at ",
ctime(), ".");
        $mail_agent:send_message(this, $system_mail_recipient,
tostr(object.name, "(", object, ") removed."), info);
      endif
    endif
  elseif (iobjstr)
    player:tell("\"", iobjstr, "\" is an unrecognized category.");
  else
    player:tell("Usage:  remove <object> from <category>");
  endif
elseif (!this:check_player(player, object))
  player:tell(this.not_control_msg);
else
  this:rm_object(object);
  player:tell("You remove ", object.name, " (", tostr(object), ").");
  if (this.public_library)
    info = tostr("Removed by ", player.name, "(", player, ") at ",
ctime(), ".");
    $mail_agent:send_message(this, $system_mail_recipient,
tostr(object.name, "(", object, ") removed."), info);
  endif
endif
"Last modified Sun Jul  6 04:32:30 1997 EDT by EricM
(#5058@DU_MainMOO).";
.

@verb #185:"preper_show" this none this "rxd"
@prog #185:preper_show
"Getting a list of object numbers.  Return a list of strings telling
something about them.";
out = {};
for i in [1..len = length(objects = args[1])]
```

```
  out = {@out, tostr(" ", i, ")",
$string_utils:space(length(tostr(len)) + 3 - (length(tostr(i)) + 1), "
"), tostr(objects[i].name))};
endfor
return $string_utils:columnize(out, 2, player:linelen());
"Last modified Tue Jun 27 16:31:20 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"find_cat" this none this "rxd"
@prog #185:find_cat
if ((ind = $string_utils:find_prefix(args[1], this.names)) != 0)
  return ind;
elseif ($string_utils:is_numeric(cat = args[1]) && ((ind = tonum(cat))
> 0 && ind <= length(this.names)))
  return ind;
else
  return 0;
endif
"Last modified Tue Jun 27 08:20:34 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"add_object" this none this "rxd"
@prog #185:add_object
if (caller != this)
  return E_PERM;
endif
pos = {@args, 1}[2];
if (!this.displays)
  this.displays = {{}};
endif
this.displays[pos] = setadd(this.displays[pos], args[1]);
"Last modified Tue Jun 27 08:02:39 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"rm_object" this none this "rxd"
@prog #185:rm_object
if (caller != this)
  return E_PERM;
endif
pos = {@args, 1}[2];
this.displays[pos] = setremove(this.displays[pos], args[1]);
"Last modified Tue Jun 27 07:59:43 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"match_disp" this none this "rxd"
@prog #185:match_disp
disp = this.displays[{@args, 1}[2]];
if ((place = tonum(subject = args[1])) != 0)
  if (place > 0 && place <= length(disp))
    return disp[place];
  endif
```

```
endif
exact_match = partial_match = {};
for object in (disp)
  if (valid(object))
    if (subject in (str_list = object.aliases))
      exact_match = setadd(exact_match, object);
    else
      for string in (str_list)
        if (index(string, subject))
          partial_match = setadd(partial_match, object);
        endif
      endfor
    endif
  endif
endfor
return length(exact_match) == 1 ? exact_match[1] | (!exact_match ?
length(partial_match) == 1 ? partial_match[1] | (!partial_match ?
$failed_match | partial_match) | exact_match);
"Last modified Wed Jun 28 04:52:44 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"@rename-cat*egory" any to any "rd"
@prog #185:@rename-category
"Usage:  @rename-cat*egory <category> to <new name>";
"Changes the name for that category.  Use this at least for the first
category, when making a single display into a multiple display.";
if (!$perm_utils:controls(player, this))
  return player:tell(E_PERM);
elseif (this.single)
  return player:tell("The room is in single category modus.");
elseif (!(cat = this:find_cat(dobjstr)))
  return player:tell(dobjstr, " not a category.");
elseif (iobjstr in this.names)
  return player:tell(iobjstr, " is already as a category name.");
else
  this.names[cat] = iobjstr;
  player:tell("Name changed.");
endif
"Last modified Wed Jun 28 12:54:37 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"help_msg" this none this "rxd"
@prog #185:help_msg
if (this.single)
  msg = this:get_message("help_single_msg");
  if (player.programmer)
    if (!this:check_player(player))
      msg = {@msg, @this:get_message("register_msg")};
    else
      msg = {@msg, @this:get_message("help_register_msg")};
    endif
  endif
```

```
else
  msg = this:get_message("help_multi_msg");
  if (player.programmer)
    if (!this:check_player(player))
      msg = {@msg, @this:get_message("register_msg")};
    else
      msg = {@msg, @this:get_message("register_multi_help_msg")};
    endif
  endif
endif
if ((check = this:get_message("help_check_msg")) && player.programmer)
  msg = {@msg, @check};
endif
if ($perm_utils:controls(player, this))
  msg = {@msg, @this:get_message("help_control_msg")};
endif
return msg;
"Last modified Wed Jun 28 15:22:30 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"get_message" this none this "rxd"
@prog #185:get_message
return typeof(msg = this.(args[1])) == STR ? {msg} | msg;
"Last modified Tue Jun 27 09:57:42 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"hidden_verbs" this none this "rxd"
@prog #185:hidden_verbs
if (caller == this || $perm_utils:controls(caller_perms(), this))
  hidden = pass(@args);
  loc = $code_utils:verb_loc();
  if (this.single || !$perm_utils:controls(args[1], this))
    hidden = setadd(hidden, {loc, "@add-cat*egory", verb_args(loc,
"@add-category")});
    hidden = setadd(hidden, {loc, "@rm-cat*egory", verb_args(loc, "@rm-
category")});
    hidden = setadd(hidden, {loc, "@rename-cat*egory", verb_args(loc,
"@rename-category")});
  endif
  if (!$perm_utils:controls(args[1], this))
    hidden = setadd(hidden, {loc, "@clear", verb_args(loc, "@clear")});
  endif
  if (!this:check_player(args[1]))
    hidden = setadd(hidden, {loc, "register", verb_args(loc,
"register")});
    hidden = setadd(hidden, {loc, "remove", verb_args(loc, "remove")});
  endif
  return hidden;
else
  return E_PERM;
endif
"Last modified Tue Jun 27 14:34:33 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
```

```
.

@verb #185:"@clear" none none none "rd"
@prog #185:@clear
"'@clear'  This will clear all displays in this room.  Be very
carefull.";
if (!$perm_utils:controls(player, this))
  return player:tell(E_PERM);
elseif (!$command_utils:yes_or_no(tostr("This will clear all displays
in ", $string_utils:nn(this), ", are you absolute sure you will do
this?")))
  return player:tell("Aborted..");
else
  this.displays = {{}};
  this.names = {"No name"};
  player:tell("Displays cleared.");
endif
"Last modified Tue Jun 27 14:53:02 1995 EDT by Kris_F (#16723@DU_Moo-
East).";
.

@verb #185:"set_category" this none this "rxd"
@prog #185:set_category
if (!this.single)
  new = args[1];
  if (index = $list_utils:iassoc(player, this.current_category))
    this.current_category[index][2] = new;
  else
    this.current_category = listappend(this.current_category, {player,
new});
  endif
endif
"Last modified Wed Jun 28 13:36:51 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"get_category" this none this "rxd"
@prog #185:get_category
if (!this.single)
  if (entry = $list_utils:assoc(player, this.current_category))
    cat = entry[2];
  else
    cat = 1;
  endif
  return cat;
endif
"Last modified Wed Jun 28 13:36:56 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"enterfunc" this none this "rxd"
@prog #185:enterfunc
pass(@args);
this:set_category(1);
```

```
"Last modified Wed Jun 28 13:37:09 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"exitfunc" this none this "rxd"
@prog #185:exitfunc
pass(@args);
if (entry = $list_utils:assoc(player, this.current_category))
  this.current_category = setremove(this.current_category, entry);
endif
"Last modified Wed Jun 28 13:37:09 1995 EDT by Ulf (#238@DU_Moo-
East).";
.

@verb #185:"more_info_for_web" this none this "rx"
@prog #185:more_info_for_web
"more_info_for_web(who, rest STR, search STR) -> html doc frg LIST";
"Returns a formatted list of objects on displayor a list of categories
available, as requested";
text = {};
rest = args[2];
search = $web_utils:url_decode(args[3]);
webcode = $web_utils:get_webcode();
if (this.single || (rest == "category" && search in this.names))
  "show listing of displayed objects";
  category = this.single ? 1 | search in this.names;
  link = tostr("<A HREF=\"&LOCAL_LINK;/", $object_browser:get_code(),
"/browse_or_edit/browse/");
  text = {@text, "<B>Objects on display:</B><UL>"};
  for item in (this.displays[category])
    if (item.icon)
      icon = tostr("<IMG SRC=\"", item:get_icon(), "\" WIDTH=32>");
    else
      icon = "";
    endif
    text = {@text, tostr("<LI>", icon, link, tonum(item), "\">",
$string_utils:nn(item), "</A>")};
  endfor
  text = {@text, "</UL>"};
  if (!this.single)
    text = {@text, tostr("<A HREF=\"&LOCAL_LINK;/", webcode, "/",
tonum(this), "#focus\">[ Show category list ]<P></A>")};
  endif
elseif (!(rest || this.single))
  "show listing of available categories";
  link = tostr("<A HREF=\"&LOCAL_LINK;/", webcode, "/", tonum(this),
"/category?");
  text = {@text, "<B>Available categories are:</B><UL>"};
  for category in (this.names)
    text = {@text, tostr("<LI>", link, $web_utils:url_encode(category),
"#focus\">", category, "</A>")};
  endfor
  text = {@text, "</UL>"};
else
```

```
  text = {@text, tostr("Sorry, this room doesn't seem to understand
what was requested of it. Please report the problem to ",
$code_utils:verb_perms().name, ".")};
endif
return text;
"Last modified Fri Nov 28 15:31:39 1997 EST by Creeper
(#101@Key_Campus).";
.

@verb #185:"init_for_core" this none this "rx"
@prog #185:init_for_core
"Copied from The System Object (#0):init_for_core by Administrator (#2)
Wed May 27 14:24:20 1998 EST";
if (caller_perms().wizard)
  pass();
  if ("server_started" in verbs(this))
    code = {"callers() || ($last_restart_time = time());"};
    set_verb_code(this, "server_started", code);
  endif
  if ("name_lookup_failed" in verbs(this))
    delete_verb(this, "name_lookup_failed");
  endif
  if ("uptime_since" in verbs(this))
    delete_verb(this, "uptime_since");
  endif
  if ("do_command" in verbs(this))
    delete_verb(this, "do_command");
  endif
  $shutdown_message = "";
  $shutdown_time = 0;
  $dump_interval = 3600;
  $gripe_recipients = {player};
  $class_registry = {{"generics", "Generic objects intended for use as
the parents of new objects", {$room, $exit, $thing, $note, $letter,
$container, $root_class, $player, $prog, $wiz, $generic_editor,
$mail_recipient, $mail_agent}}, {"utilities", "Objects holding useful
general-purpose verbs", {$string_utils, $gender_utils, $trig_utils,
$time_utils, $match_utils, $object_utils, $lock_utils, $list_utils,
$command_utils, $code_utils, $perm_utils, $building_utils}}};
  set_verb_code(this, "user_connected", verb_code(this,
"user_connected(core)"));
  delete_verb(this, "user_connected(core)");
endif
.

----- End of @archive #185
```

$connection

```
@archive #318 with verbs and properties
----- Beginning of @archive #318

@create #213 named Connection Prototype
;#318.("active") = 0
;#318.("entrances") = {#313}
```

```
;#318.("blessed_object") = #-1
;#318.("blessed_task") = 1996376525
;#318.("exits") = {#305}
;#318.("invited") = {}
;#318.("instruction") = {"Use this prototype to connect rooms. Exits
can be added with @add-exit, and will be showed with the description.",
"You can program timed events, using scripts. See help @scripts.", "In
this area, you can set an atmosphere and other little tricks. See help
here for further inforamtion."}
;#318.("prototype") = #318
;#318.("key") = 0
;#318.("aliases") = {"Connection Prototype"}
;#318.("description") = {"This is the prototype for a connection
area.", "You can use a child of this if you want to build, for example,
a corridor, or a lift, or a hall, or a place that leads to other
places."}
;#318.("object_size") = {5664, 896860816}
;#318.("web_calls") = 26
;#318.("last_modified") = 875164631
;#318.("creation_date") = 875164631

@verb #318:"init_for_core" this none this "rx"
@prog #318:init_for_core
"Copied from The System Object (#0):init_for_core by Administrator (#2)
Thu Sep 25 16:56:29 1997 EST";
if (caller_perms().wizard)
  pass();
  if ("server_started" in verbs(this))
    code = {"callers() || ($last_restart_time = time());"};
    set_verb_code(this, "server_started", code);
  endif
  if ("name_lookup_failed" in verbs(this))
    delete_verb(this, "name_lookup_failed");
  endif
  if ("uptime_since" in verbs(this))
    delete_verb(this, "uptime_since");
  endif
  if ("do_command" in verbs(this))
    delete_verb(this, "do_command");
  endif
  $shutdown_message = "";
  $shutdown_time = 0;
  $dump_interval = 3600;
  $gripe_recipients = {player};
  $class_registry = {{"generics", "Generic objects intended for use as
the parents of new objects", {$room, $exit, $thing, $note, $letter,
$container, $root_class, $player, $prog, $wiz, $generic_editor,
$mail_recipient, $mail_agent}}, {"utilities", "Objects holding useful
general-purpose verbs", {$string_utils, $gender_utils, $trig_utils,
$time_utils, $match_utils, $object_utils, $lock_utils, $list_utils,
$command_utils, $code_utils, $perm_utils, $building_utils}}};
  set_verb_code(this, "user_connected", verb_code(this,
"user_connected(core)"));
  delete_verb(this, "user_connected(core)");
endif
.
```

```
----- End of @archive #318
```

$building

```
@archive #531 with verbs and properties
----- Beginning of @archive #531

@create #185 named Building Prototype
;#531.("current_category") = {}
;#531.("entrances") = {#379}
;#531.("blessed_object") = #-1
;#531.("blessed_task") = 859465688
;#531.("exits") = {#281}
;#531.("invited") = {}
;#531.("prototype") = #531
;#531.("key") = 0
;#531.("aliases") = {"Building Prototype"}
;#531.("description") = {"This is a prototype for a building.", "You
can use this to define a whole building, like for example, a post
office, or a Faculty building.", "Add exits and entrances which lead to
the building rooms, and use the directory to explain their
destination."}
;#531.("object_size") = {3269, 896860816}
;#531.("web_calls") = 17
;#531.("last_modified") = 875164638
;#531.("creation_date") = 875164638

@verb #531:"init_for_core" this none this "rx"
@prog #531:init_for_core
"Copied from The System Object (#0):init_for_core by Administrator (#2)
Thu Sep 25 16:56:10 1997 EST";
if (caller_perms().wizard)
  pass();
  if ("server_started" in verbs(this))
    code = {"callers() || ($last_restart_time = time());"};
    set_verb_code(this, "server_started", code);
  endif
  if ("name_lookup_failed" in verbs(this))
    delete_verb(this, "name_lookup_failed");
  endif
  if ("uptime_since" in verbs(this))
    delete_verb(this, "uptime_since");
  endif
  if ("do_command" in verbs(this))
    delete_verb(this, "do_command");
  endif
  $shutdown_message = "";
  $shutdown_time = 0;
  $dump_interval = 3600;
  $gripe_recipients = {player};
```

```
  $class_registry = {{"generics", "Generic objects intended for use as
the parents of new objects", {$room, $exit, $thing, $note, $letter,
$container, $root_class, $player, $prog, $wiz, $generic_editor,
$mail_recipient, $mail_agent}}, {"utilities", "Objects holding useful
general-purpose verbs", {$string_utils, $gender_utils, $trig_utils,
$time_utils, $match_utils, $object_utils, $lock_utils, $list_utils,
$command_utils, $code_utils, $perm_utils, $building_utils}}};
  set_verb_code(this, "user_connected", verb_code(this,
"user_connected(core)"));
  delete_verb(this, "user_connected(core)");
endif
.

----- End of @archive #531
```

$mobile

```
@archive #881 with verbs and properties
----- Beginning of @archive #881

@create #254 named Mobile Area Prototype
;#881.("inside_description") = {"A silent place.", "A small array of
glittering buttons seems to vanish as you try to touch it.", "Inside
here looks a lot like a dangerous place.", "You can navigate in the MOO
using the 'fly' command.", "See the help for more information."}
;#881.("active") = 0
;#881.("entrances") = {#882}
;#881.("blessed_object") = #-1
;#881.("blessed_task") = 955260146
;#881.("exits") = {#883}
;#881.("invited") = {}
;#881.("key") = 0
;#881.("aliases") = {"Mobile Area Prototype"}
;#881.("description") = {"It looks like something which can start
moving at any time.", "It is very silent here, and some glowing buttons
can be seen in front of you, but as you try to touch them, they
disappear."}
;#881.("object_size") = {6176, 896860816}
;#881.("web_calls") = 24
;#881.("last_modified") = 892790119
;#881.("creation_date") = 892790119

@verb #881:"init_for_core" this none this "rx"
@prog #881:init_for_core
"Copied from The System Object (#0):init_for_core by Administrator (#2)
Fri Apr 17 15:16:45 1998 EST";
if (caller_perms().wizard)
  pass();
  if ("server_started" in verbs(this))
    code = {"callers() || ($last_restart_time = time());"};
    set_verb_code(this, "server_started", code);
  endif
  if ("name_lookup_failed" in verbs(this))
    delete_verb(this, "name_lookup_failed");
  endif
```

```
  if ("uptime_since" in verbs(this))
    delete_verb(this, "uptime_since");
  endif
  if ("do_command" in verbs(this))
    delete_verb(this, "do_command");
  endif
  $shutdown_message = "";
  $shutdown_time = 0;
  $dump_interval = 3600;
  $gripe_recipients = {player};
  $class_registry = {{"generics", "Generic objects intended for use as
the parents of new objects", {$room, $exit, $thing, $note, $letter,
$container, $root_class, $player, $prog, $wiz, $generic_editor,
$mail_recipient, $mail_agent}}, {"utilities", "Objects holding useful
general-purpose verbs", {$string_utils, $gender_utils, $trig_utils,
$time_utils, $match_utils, $object_utils, $lock_utils, $list_utils,
$command_utils, $code_utils, $perm_utils, $building_utils}}};
  set_verb_code(this, "user_connected", verb_code(this,
"user_connected(core)"));
  delete_verb(this, "user_connected(core)");
endif
.

----- End of @archive #881
```

Families

```
@family #3
Ancestors/descendants information on generic room (#3):
{30} Root Class (#1) ..............................: Administrator (#2)
  {4} generic room (#3) ............................: Administrator (#2)
    {5} Generic Editor (#49) .............................: Hacker (#36)
      Verb Editor (#48) ..................................: Hacker (#36)
      {1} Note Editor (#47) ..............................: Hacker (#36)
        Section Editor (#441) ............................: Hacker (#36)
      Mail Room (#46) ....................................: Hacker (#36)
      HTML Editor (#137) ..........................: BioGate-owner (#113)
      Design Studio (#1048) ..............................: Creeper (#101)
    {2} Generic Improved Room (#184) .................: Administrator (#2)
      {4} Generic Improved Room with Cleaning ++ (#206) ...: Administrator (#2)
        Model Improved Room with Cleaning and Scripts (#237) ....: Owner (#107)
        {13} Social Area Prototype (#213) .............: Administrator (#2)
          {3} Generic Improved Portable Room (#216) ..: Administrator (#2)
            {1} Generic Improved Tardis (#217) .......: Administrator (#2)
              {1} Tardis Portable Room (#254) ..............: Owner (#107)
                {3} Mobile Area Prototype (#881) .....: Administrator (#2)
                  the Flying Carpet (#514) .................: Owner (#107)
                  lunchbox (#544) ...........................: anti (#540)
                  wafting clouds (#717) ...................: anmore (#284)
            {1} Basic Portable Room (#253) .................: Owner (#107)
              a globe (#888) ..............................: Creeper (#101)
            DU Portal (#869) ..............................: Creeper (#101)
          Model Lively Room (#246) ........................: Owner (#107)
          {10} Connection Prototype (#318) ...........: Administrator (#2)
            Classrooms (#257) .............................: Owner (#107)
            The Hall (#61) ............................: Administrator (#2)
            Office Area (#188) ............................: Owner (#107)
            Resources Room (#572) ...................: Administrator (#2)
```

```
    Professional Area (#473) .....................: Owner (#107)
    Staff Rooms (#258) ...........................: Owner (#107)
    Student Rooms (#435) .........................: Owner (#107)
    Prototypes Area (#521) ...................: Administrator (#2)
    The Pit (#993) ...............................: johnb (#757)
    LINKS (#999) .................................: dinesh (#751)
  The Sneephole (#653) ...................: sneep-the-beep (#182)
  The Green Lake (#191) ........................: Creeper (#101)
  The Studio (#821) ............................: Rich (#615)
  Meeting Room (#198) ..........................: Owner (#107)
  anti-room (#541) .............................: anti (#540)
  The Word (#354) ..............................: anmore (#284)
  Alive! (#452) ................................: Creeper (#101)
  LendLease Area (#498) ........................: nick (#306)
  park1 (#925) .................................: ken1 (#754)
  Foyer of Johnb's place (#771) ..................: johnb (#757)
{7} Learning Area Prototype (#211) ...........: Administrator (#2)
  Model Classroom (#243) .......................: Owner (#107)
  Theory and Practice Room (#528) ..............: Owner (#107)
  Computer-Based Design Room (#632) ............: Owner (#107)
  AI in Design (#436) ..........................: Owner (#107)
  Communications Elective (#598) ...............: Owner (#107)
  mary's study (#351) ..........................: Mary (#297)
  Dave's Temporary Testing Chamber (#104) ..........: Dave (#100)
{24} Private Room Prototype (#434) ...........: Administrator (#2)
  Tea Room (#558) ..............................: nick (#306)
  Laboratory (#626) ............................: Brad (#226)
  Eric's Nook (#576) ...........................: EricM (#94)
  Testoffice (#689) ............................: nick (#306)
  Generic Office (#470) ........................: nick (#306)
  Nick's Design Lab (#475) .....................: nick (#306)
  Creeper's PhD Design Studio (#195) ...........: Creeper (#101)
  {39} Office Prototype (#693) .............: Administrator (#2)
    brett's office (#826) ......................: brett (#745)
    Toni's office (#410) .......................: Toni (#672)
    mixer (#392) .............................: Creeper (#101)
    Brad's Office (#231) .......................: Brad (#226)
    Fyock's Office (#276) ......................: Fyock (#223)
    Tim's Office (#659) ..........................: Tim (#617)
    x-15 (#307) ..............................: Gerard (#668)
    1226's Office (#496) .......................: 1226 (#418)
    The Mouse Hole (#479) ......................: hong (#340)
    jeya's office (#448) .......................: jeya (#304)
    DavidS's Office (#394) .....................: davidS (#286)
    Wingy's Tent (#535) ........................: Wingy (#534)
    Joe's Office (#384) ..........................: Joe (#332)
    Moto's Office (#443) .......................: moto (#335)
    phibbsie's office (#731) ..................: Phibbsie (#734)
    Mary's Office (#221) .......................: Mary (#297)
    The Inner Sanctum (#592) ...................: Brad (#226)
    Nick's Office (#308) .......................: nick (#306)
    Jane's Office (#278) .......................: jane (#341)
    MotMot's Web (#449) ........................: motmot (#383)
    Raul's Office (#353) .......................: Raul (#311)
    Rich's sanctum (#621) ......................: Rich (#615)
    Tim's Workshop (#629) ........................: Tim (#617)
    Jose's Place (#396) ........................: jose (#342)
    Anmore's Office (#323) .....................: anmore (#284)
    The Primal Chaos (#891) ....................: Janus (#893)
```

```
            Kantor Jokse (#801) ..........................: Jokse (#755)
            Gil's cyberoffy (#397) .......................: gil (#760)
            design lab (#772) ............................: dinesh (#751)
            studio (#892) ................................: neha (#759)
             space  (#909) ...............................: neha (#759)
            Designer's Heaven (#913) .....................: boss (#756)
            Oasis (#916) .................................: boss (#756)
            Tea House (#965) .............................: tony (#762)
            The Pad (#1002) ..............................: johnb (#757)
            dunny (#1009) ................................: Blade (#743)
            Scott's Office (#829) ........................: Scott (#767)
            Pol's Palace (#1034) .........................: pol (#770)
            Office ONION (#244) ..........................: tommy (#753)
          study1 (#897) ..................................: ken1 (#754)
          Jokse'sMusikRoom (#588) ........................: Jokse (#755)
          Jane's Office (#741) ...........................: janec (#747)
          studio qui (#509) ..............................: qui (#863)
          The Lair (#337) ..............................: alleycat (#746)
          The Chamber (#948) .............................: Blade (#743)
          tory's place (#921) ............................: Tory (#763)
          Eddie's_Virtual_Studio (#766) ..................: Eddie (#744)
          hung's virtual design studio (#959) ............: hung (#749)
          Stevie's retreat (#414) ........................: steve (#761)
          dark's_office (#768) ...........................: dark (#752)
          The Pit (#998) .................................: johnb (#757)
          Geoff's Workroom (#679) ........................: geoff (#748)
          private box (#850) .............................: gm (#750)
          living area (#1038) ............................: Eddie (#744)
          pop_s office (#794) ............................: pop (#758)
      Model Improved Room (#234) .........................: Owner (#107)
    Model Portable Rooms (#249) ..........................: Owner (#107)
    {4} Storage Area Prototype (#185) ................: Administrator (#2)
      {2} Building Prototype (#531) ..................: Administrator (#2)
        Research Dungeon (#905) ..........................: dinesh (#751)
        Boss' Oasis (#525) ...............................: boss (#756)
      Objects Library (#201) .............................: Owner (#107)
      Documents Library (#562) .......................: Administrator (#2)
      herb garden (#856) ...............................: alleycat (#746)
* 1 ancestor, 124 descendants found.
```

APPENDIX E. LambdaMOO *B:Arbitration

Arbitration (#50392)
---
by Grump (#122)
[Last edited before the system kept track of the time]

This petition creates a method for resolving disputes in LambdaMOO. Most of the 'social problems' in LambdaMOO can be seen as disputes between players: the conflict of one player's rights to expression against other player's rights to freedom from harrassment, conficts over building, RPG, or whatever. By this petition:

* Create a registry of volunteer arbitrators: MOOers who volunteer to listen to the various sides in a dispute and decide on an appropriate resolution. Anyone can volunteer to be an arbitrator, subject to a few qualifications listed below.

* Anyone involved in a dispute can explicitly call for 'arbitration', and name (only) one other party in the dispute.

* Other players can join in a dispute if they want, but they don't get to choose the arbitrator. They're named to make sure their point of view is heard.

* The original disputants have a limited period of time (48 hours) to AGREE on an arbitrator for their case, from among arbitrators who agree to volunteer to arbitrate the case. If those involved in a dispute cannot agree on an arbitrator for their case, one will be chosen randomly for them from among arbitrators who had volunteered to hear the case within the 48 hour period.

* The arbitrator for a case has the power to call for almost any action WITHIN THE MOO that is available, insofar as it only affects those involved in the dispute. For example, this would include @toading or @newting a character for any length of time, modifying their quota, revealing their site information, recycling objects, etc. Arbitrators can decide to not act at all, or to banish all participants in the dispute, etc. The arbitrator can judge the person who 'called' the dispute as well, e.g., if the dispute was spurious.

* The arbitrator to a case can have access to all site and connect information of the parties involved in the dispute. Such access is logged.

* Once an arbitrator is chosen and arbitration begins, the arbitrator should try to hear all the sides of the dispute, but not delay deciding unreasonably. Arbitrators are encouraged, but not required, to solicit advice on the handling of the case from others. It's up to the arbitrator to schedule how they hear the case, what process they follow.

* If someone invokes a dispute and the other party doesn't respond, (for whatever reason: they haven't logged on, refuse to cooperate, don't participate, etc.) the arbitrator can go ahead and make some decision in their absence. Such decisions should be reasonable for the circumstances. (Presumably, this will be a 'randomly chosen' arbitrator, since the other party didn't respond.)

* It is possible to have a dispute with a guest. In that case, the dispute is collectively with 'all guests that come from that site'.

* The arbitrator has to write a summary of the case, what they heard, and what their decision is, and post it to a public place.

* An arbitration will 'time out' and be cancelled if no decision is made within a reasonable period of time. (2 weeks).

* There will be a delay (24 hour) between posting of a decision and the carrying out of whatever action it contains. All actions are logged in a public place.

* There are some qualifications for being an arbitrator. These include:

- Having connected to LambdaMOO for a while (4 months).
- Of course, must volunteer for the job, not only 'in general' but for the specific arbitration.
- No obvious conflict of interest (in particular, cannot come from the same site as any of the participants in the dispute).
- Have registered as an arbitrator for a while before the dispute is called (2 weeks).

The 'player' refers to a primary character; you shouldn't be able to beat the system and judge yourself by faking multiple characters.

While arbitrators decisions are not subject to appeal, they are subject to review: if any five MOOers (otherwise qualified to be arbitrators) vote to do so, they can overturn a judgement. In addition, if a judgement is overturned, and ten additional qualified MOOers vote to do so, the arbitrator whose judgement was overturned will be barred from arbitrating again. (This is primarily a safeguard against them being chosen randomly again.) The same conflict of interest restrictions apply for voting here as apply for arbitrators deciding cases. Judgements can be overturned because they are too harsh, or the arbitrator has a serious confict of interest but still didn't disqualify themselves, etc., or even because it isn't implementable.

Minor elements of this petition can be changed by a simpler mechanism than another petition: by a vote of 30 qualified MOOers, new qualifications for abitrators can be added, or any of the numbers of thresholds, time limits can be changed, or minor changes made to the procedure to add or shorten the delays.

This petition leaves out implementation details; initially, it could be handled by mail, enforcement could be handled by arbitrators asking the wizards to perform various actions; but eventually, much of it can be automated, with some care to avoid abuse and make sure records are kept.

This petition also doesn't contain any guidelines as to proper rules of behavior; it is presumed that arbitrators will use good judgement, and their understanding of common sense and manners. Arbitrators are expected to respond with the smallest action necessary to resolve the dispute, and the review process is intended as a way to insure that it happens that way.

APPENDIX F. Petition #7976 and relative messages

This petition was proposed on LambdaMOO in December 1997. Follows the text of the petition and relative messages of comment.

```
                        RoomDescriptions (#7976)
                        ------------------------
                          by creeper (#106368)
              [Last edited on Thursday, December 18, 1997 at 2:04 pm]

   This proposal is for the implementation of a new system to allow
   qualified players (designers) to control and regulate the design of the
   house.

   PREMISES
   - The @describe command is the only design tool available in the MOO.
   - Certain players should be able to modify rooms descriptions, ie
   design them.
   - The Design capabilities should be given to experienced players, as
   the arbitrators are, elected in the Design Board, via public
   candidatures and elections.
   - The buidling of new public rooms should pass the Design Board vote
   and fulfil Design specifications

   IMPLEMENTATION
   - a new class of players: designers
   - a new set of properties for all the room herarchy to allow them to be
   'designable' by the above class of players
   - a new set of commands (@design, @sketch) which allow the designer
   class to work more easily
   - a set of design specifications should be written and they should work
   similarly to the ballot/petition system
   - a Design Board, whose components will revise design projects for new
   public rooms, and major changes to the spaces in the house

   Note that this proposal could represent a major step in the MOO
   development, focusing on the Design aspects of a virtual
   place/reality/world.
   However, citizens would be still free to design their own spaces and
   objects.

   Message 1 on *Petition:Room_Descriptions (#7976):
   Date:    Thu Dec 11 21:49:23 1997 PST
   From:    NightAngel (#90845)
   To:      *Petition:Room_Descriptions (#7976)
```

You know someone's going to ask it, so it might as well be me: How do you know which rooms are "public"?

Message 2 on *Petition:Room_Descriptions (#7976):
Date: Thu Dec 11 21:50:22 1997 PST
From: Gear (#104262)
To: *Petition:Room_Descriptions (#7976)

How exactly will this command modify the description of the rooms?

Message 3 on *Petition:Room_Descriptions (#7976):
Date: Thu Dec 11 21:59:18 1997 PST
From: creeper (#106368)
To: NightAngel (#90845) and *Petition:Room_Descriptions (#7976)

> You know someone's going to ask it, so it might as well be me: How
> do you know which rooms are "public"?

The ones which are "navigable", like #17, or the dining, or the garage, and so on. Basically any room, which is not hidden or private (like my own rooms, for example). I think that anyone should design by their own words the new lambdahouse.

Message 4 on *Petition:Room_Descriptions (#7976):
Date: Thu Dec 11 22:01:13 1997 PST
From: creeper (#106368)
To: Gear (#104262) and *Petition:Room_Descriptions (#7976)

>How exactly will this command modify the description of the rooms?

I'd say that it works like @describe, only it affect another property which coud be called by :look_self. So the main info, like special instructions, or exits, can be maintained, and only the "design" would be changed.

Message 5 on *Petition:Room_Descriptions (#7976):
Date: Sat Dec 13 05:13:01 1997 PST
From: Yib (#58337)
To: *Petition:Room_Descriptions (#7976)
Subject: Everybody plays

So the database would potentially have to store as many living room descriptions as players who use it, the dining room would potentially have to store as many dining room descriptions as players who use it, the driveway would potentially have to store as many driveway descriptions as players who use it, and The Sex Room wouldn't, because it's not walkably connected (or maybe it is, and then there's more bloat).

I don't think the database could support it. Or maybe I misunderstand what this petition is getting at?

--Yib

Message 6 on *Petition:Room_Descriptions (#7976):
Date: Sat Dec 13 10:57:26 1997 PST

From: Gary (#110811)
To: *Petition:Room_Descriptions (#7976)
Subject: Interpreting the Petition

The petition now says:
> Registered users should be able to modify the room descriptions of public spaces.
> Using the @describe command, is the only way of designing, and it should be possible to allow MOOers to redesign areas of the house, not just rooms they own.
> I propose the implementation of a command (@sketch, or @design) which allows users to modify the description of a room, without changing any substantial information, such as the exits, or the objects contained or any other instruction related to the space.
> All that information can be contained in a separate property (and recalled by :look_self).

Uhh, it sounds to me like the petitioner wants to make the .description property on 'public' rooms writeable by any non-guests, leaving such things as contents and exits unchanged of course.
Yib reads it to mean that each non-guest could create a different room desc that they alone see. That's actually something that could be accomplished to some extent by making a player class that had its own look verb and allowed each player to save a personal desc for *any* rooms. That would also put the quota burden on the individual players who want this feature. Hmm, I guess it couldn't change the desc that scrolls up when you first enter the room though. Anyway, while I *think* I agree with Yib that this doesn't sound like such a good idea, at least the player class approach would be better than just letting any schmuck rewrite the Living Room desc. Gawd, I can see 'LR Desc Wars' would be next.
It's an interesting idea though, creeper. No personal criticism is intended by me. Keep on writing.

Message 7 on *Petition:Room_Descriptions (#7976):
Date: Sat Dec 13 17:29:15 1997 PST
From: Yib (#58337)
To: *Petition:Room_Descriptions (#7976)

You wanna redecorate a quote-public-thing-unquote? Reupholster the couch! :)

Message 8 on *Petition:Room_Descriptions (#7976):
Date: Sun Dec 14 17:55:22 1997 PST
From: creeper (#106368)
To: *Petition:Room_Descriptions (#7976)
Subject: Re: Interpreting the Petition

> Uhh, it sounds to me like the petitioner wants to make the
> .description property on 'public' rooms writeable by any
> non-guests, leaving such things as contents and exits unchanged of
> course.

Not the .description, but another property called by look_self.
The .description should contain some info regarding the room (unchangeable), such as exits, other directions, particular instructions.

I didn't suggest that a room should have as many descriptions as players (or nearly), but that each player should be able to change it.
Maybe creating a new player class "designer" is a good idea.
Changing the description only for a single player (say, I will have a creeper's description of #17), is not a good idea, for a lot of interaction between players happens thanks to the room description (so all the people in the room should see the same thing).

I see that it's difficult to handle the 'creativity' of everybody, for they would like to rewrite the description lots of times, and maybe in a non acceptable way....
So, as I see it, a new class of players, designers, can allow people to re-design designable rooms.....
We could have rooms which are designable, and only designers can design them, and other rooms which are not. So, we would create a new class of rooms, and a new class of players. And, ultimately, a set of verbs for design.
Brainstorming continues.

Message 9 on *Petition:Room_Descriptions (#7976):
Date: Sun Dec 14 20:19:25 1997 PST
From: Gear (#104262)
To: *Petition:Room_Descriptions (#7976)
Subject: New Text

Would this new version of the petition affect public rooms? If not, why does it need a petition? And if it does, how do you determine who gets to be a designer? A test of creativity? An oath to only make themely designs? This new version is better, but still needs some work.

Message 10 on *Petition:Room_Descriptions (#7976):
Date: Mon Dec 15 01:40:27 1997 PST
From: creeper (#106368)
To: *Petition:Room_Descriptions (#7976)
Subject: Re: New Text

> Would this new version of the petition affect public rooms? If

It would affect public rooms.

> not, why does it need a petition? And if it does, how do you
> determine who gets to be a designer? A test of creativity? An

The class of player should be free (anybody can change to it), but public rooms would need to carry a new prop (the one which is changeable).

I agree that I need some more understanding of how/who/where/what.
As I see it now:
- players who are designers (who belong to the player class designer) can change a property in all rooms (unless anti-designer flag is set), and certainly in all the public rooms, ie all the ones which are able to be accessed simply walking around the house.
- rooms can either be designable (all the public, and whoever chooses to make one's own), or not

- a property .design (or similia) would store the info given by designers

That's mine, so far....

Message 11 on *Petition:Room_Descriptions (#7976):
Date: Tue Dec 16 18:54:20 1997 PST
From: Pug (#92528)
To: *Petition:Room_Descriptions (#7976)
Subject: Question

I have a few questions: When someone makes a modification to a room, does it only affect the description given to them, or to everyone else? If it was only to them, I don't think you need a petition since the individual is affected, not the public. If it is for everyone to see... then:
Who decides who gets to be a designer? Obviously you don't want 500 odd designers running around the place starting descriptionwars, trashing the Lambda house do we? It would be cool to have more interactivity, but it needs some careful planning and some checks-and-balances.
Unfortunately, with the apparent possibility for a 2-day old char to become a designer and promptly paint the entire Living Room black and knock out the lightbulbs doesn't paint a pretty picture, excuse the terrible pun. It would be cool if themes of certain rooms, including public ones, were changed by 'in-house' artists, who are chosen by the owners of the rooms. For public rooms, if you wanted, you could hold a direct election, or elect an interior design committee who would then choose an artist. Just as in real life, I suppose, except cheaper :)

Pug.

Message 12 on *Petition:Room_Descriptions (#7976):
Date: Thu Dec 18 16:52:50 1997 PST
From: creeper (#106368)
To: Gary (#110811), *Petition:Room_Descriptions (#7976), Gear (#104262), Pug (#92528), Brack (#90845), GoneButNotForgotten (#58337), and anmore (#112154)
Subject: Petition #7976

#7976 rewritten
I think we can do something with it.
Thanx for ur interest.

Message 13 on *Petition:Room_Descriptions (#7976):
Date: Fri Dec 19 07:07:40 1997 PST
From: Daydreamer (#91923)
To: *Petition:Room_Descriptions (#7976)

Actually, I like this idea... but not that everyone should be able to change the rooms. The rooms should be stable... how about this...

Have "Designer" be an electable post, such as the Reapers. Elect five or so, and give them run of LambdaHouse... they'll take care of linking in rooms, maintaining descriptions, etc. As I see it, the current problem is that LambdaHouse has no one "owner" (or group of maintainers) who works on linking things up and maintaining.

How would I define "LambdaHouse"? Probably all the things that you can walk to, and are actually part of the "House" complex... this includes the #17 area, outside around the pool, the basement/furnace room, etc. Not, for instance, The Looking Glass Bar, or anything not owned by one of the original creators.

Any help here? :)

Message 14 on *Petition:Room_Descriptions (#7976):
Date: Fri Dec 19 07:09:13 1997 PST
From: Daydreamer (#91923)
To: *Petition:Room_Descriptions (#7976)

Revision to last post: Perhaps everything owned by Lambda (#50))?

Message 15 on *Petition:Room_Descriptions (#7976):
Date: Mon Dec 22 11:56:57 1997 PST
From: Brack (#90845)
To: *Petition:Room_Descriptions (#7976)

Even Lambda himself?

Message 16 on *Petition:Room_Descriptions (#7976):
Date: Mon Dec 22 12:01:28 1997 PST
From: Daydreamer (#91923)
To: *Petition:Room_Descriptions (#7976)

Ahem. I meant everything in the HOUSE owned by Lambda. It was just an idea, anyway. :P

Message 17 on *Petition:Room_Descriptions (#7976):
Date: Mon Dec 22 14:11:55 1997 PST
From: creeper (#106368)
To: *Petition:Room_Descriptions (#7976), *NewSocialIssues (#17671), *projects (#44433), and *Research (#9420)
Subject: design in the house

> Have "Designer" be an electable post, such as the Reapers. Elect
> five or so, and give them run of LambdaHouse... they'll take care
> of linking in rooms, maintaining descriptions, etc. As I see it,
> the current problem is that LambdaHouse has no one "owner" (or
> group of maintainers) who works on linking things up and
> maintaining.

yes, i agree in creating a 'design board' and creating a design system by which the moo designers take care of the house.
we wouldn't need to have any owner of the house to change the descriptions.
i reckon that the designers should take care of the public spaces, defined as the "walkable" ones.
i think that if #7976 passes, it'd create VERY interesting results.

Message 18 on *Petition:Room_Descriptions (#7976):
Date: Mon Dec 22 16:01:42 1997 PST

```
From:    *Petition:Room_Descriptions (#7976)
To:      *Wizard-List (#6428), creeper (#106368), and
*Petition:Room_Descriptions (#7976)
Subject: Request for vetting
Reply-to: creeper (#106368), *Wizard-List (#6428), and
*Petition:Room_Descriptions (#7976)
Sender:   Petitioner (#4)

creeper, the author of Petition:Room_Descriptions (#7976):
`RoomDescriptions', has acquired 10 signatures on his/her petition and
is submitting it to you, the wizards, for vetting.  Please look it over
and either
   1) type `approve #7976' to grant it your mark of approval
or 2) type `deny #7976' to refuse such approval and then send mail to
*Petition:Room_Descriptions explaining your reasons for doing so.

Thank you for your attention to this matter.
```

APPENDIX G. Links to Electronic Sites

Papers

Cherny, Lynn. (1995a) *The Modal Complexity of Speech Events in a Social MUD.* (http://www.research.att.com/~cherny/ejc.txt)

Cherny, Lynn. (1995b) *The MUD Register: Conversational Models of Action in a Text-Based Virtual Reality*. PhD Dissertation, Stanford University. (http://www.research.att.com/~cherny/diss-overview.html)

Cicognani, Anna. (1996) "Which language for Cyberspace? (poster)." (http://www.arch.usyd.edu.au/~anna/images/CVEposter.jpg)

Cicognani, Anna. Ed. (1997) *Creative Collaboration in Virtual Communities (VC97*) (http://www.arch.usyd.edu.au/kcdc/conferences/VC97/)

Cicognani, Anna. (1998) "On the linguistic nature of Cyberspace and Virtual Communities." (http://www.arch.usyd.edu.au/~anna/papers/language.pdf)

Cicognani, Anna and Maher, Mary Lou. (1997) "Models of Collaboration for Designers in a Computer-Supported Environment." (http://www.arch.usyd.edu.au/~anna/papers/ifip97a.html)

Clarke-Willson, Stephen. (1998) *Applying Game Design to Virtual Environments.* (http://www.gamasutra.com/features/game_design/980101/virtual_environments1.htm)

Curtis, Pavel. (1992) *On to the next stage...* (http://vesta.physics.ucls.edu/~smolin/lambda/laws_and_history/newdirection)

Curtis, Pavel. (1996) *LambdaMOO Programmer's Manual. For LambdaMOO Version 1.8.0p5*. (ftp://dulles.placeware.com/pub/MOO/ProgrammersManual.html)

Curtis, Pavel and Nichols, David A. (1993) *MUDs grow up: Social Virtual Reality in the Real World*. (ftp://parcftp.xerox.com/pub/MOO/papers/MUDsGrowUp.txt)

Danet, Brenda. Ed. (1995) *Play and Performance in Computer-Mediated Communication.* (http://www.ascusc.org/jcmc/vol1/issue2/tocon.html)

Davis, Stephen Boyd, Huxor, Avon and Lansdown, John. (1996) *The Design of Virtual Environments, with particular reference to VRML*. (http://www.man.ac.uk/MVC/SIMA/vrml_design/title.html)

Dibbell, Julian. (1993) “Rape in Cyberspace or how an evil clown, a haitian trickster spirit, two wizards, and a cast of dozens turned a database into a society.” (ftp://dulles.placeware.com/pub/MOO/papers/VillageVoice.txt)

Fernback, Jan and Thompson, Brad. (1995) *Virtual Communities: Abort, Retry, Failure?* (http://www.well.com/user/hlr/texts/VCcivil.html)

Gay, Geri and Lentini, Marc. (1995) “Use of Communication Resources in a Networked Collaborative Design Environment.” (http://www.ascusc.org/jcmc/vol1/issue1/IMG_JCMC/ResourceUse.html)

Gero, John S. (1990) “Design prototypes: a knowledge representation schema for design.” (http://www.arch.usyd.edu.au/~john/publications/ger-prototypes/ger-aimag.html)

Hill, Belinda. (1996) *Voices from Cyberspace: the Metaphors of Electronic Communication*. (http://weber.u.washington.edu/~belinda/voices.html)

JCMC. (1996) *Journal of Computer-Mediated Communication*. (http://shum.cc.huji.ac.il/jcmc/jcmc.html)

Kolko, Beth E. (1995) “Building a World with Words: The Narrative Reality of Virtual Communities.” (http://acorn.grove.iup.edu/en/workdays/Kolko.html)

Kollock, Peter and Smith, Marc. (1994) “Managing the Virtual Commons: Cooperation and Conflict in Computer Communities.” (http://netscan.sscnet.ucla.edu/csoc/papers/virtcomm/vcommons.htm)

Kvan, Thomas. (1994) “Reflections on computer-mediated architectural design.” (http://arch.hku.hk:80/people/tkvan/acm-94.html)

Lombard, Matthew and Ditton, Theresa. (1997) *At the Heart of It All: The Concept of Presence*. (http://www.ascusc.org/jcmc/vol3/issue2/lombard.html)

Maher, Mary Lou, Cicognani, Anna and Simoff, Simeon. (1996) “An experimental study of computer-mediated collaborative design.” (http://www.arch.usyd.edu.au/kcdc/cmcd/paper/)

Maher, Mary Lou, Simoff, Simeon and Cicognani, Anna. (1997) “Observations from an Experimental Study of Computer-Mediated Collaborative Design.” (http://www.arch.usyd.edu.au/kcdc/cmcd/paper/ifip97b.pdf)

McCarthy, John. (1996) *Elephant 2000: A Programming Language Based on Speech Acts*. (http://www-formal.stanford.edu/jmc/elephant/elephant.html)

Mitchell, William. (1995a) *Commentary*. (http://www.feedmag.com/95.08dialog/95.08mit2.html)

Mnookin, Jennifer. (1996) "Virtual(ly) Law: The Emergence of Law in LambdaMOO." (http://www.ascusc.org/jcmc/vol2/issue1/lambda.html)

Mynatt, Elizabeth D. et al. (1997) "Design for Network Communities." (http://www.acm.org/isgchi/chi97/proceedings/edm.htm)

Paccagnella, Luciano. (1997) "Getting the Seats of your Pants Dirty: strategies for Ethnographic Research on Virtual Communities." (http://www.ascusc.org/jcmc/vol3/issue1/paccagnella.html)

Reid, Elizabeth. (1994) *Cultural Formations in Text-Based Virtual Realities.* (http://people.we.mediaone.net/elizrs/work.html)

Reid, Elizabeth M. (1991) *Electropolis. Communication and Community on Internet Relay Chat.* (http://people.we.mediaone.net/elizrs/work.html)

Rohrer, Tim. (1997) *Conceptual Blending on the Information Highway: How Metaphorical Inferences Work*. (http://wwwdarkwing.uoregon.edu/~rohrer/iclacnf4.htm)

Self, John. (1995) *Computational Mathetics: Towards a Science of Learning Systems Design*. (http://www.cbl.leeds.ac.uk/~jas/cm.html)

Stivale, Charles J. (1995) *help manners: Frontier Tales of Two MOOs*.

(http://wwwpub.utdallas.edu/~cynthiah/lingua_archive/help_manners.html)

Other links:

The B-Org: http://www.the-b.org

Dictionary of Linguistics: http://wwwots.let.run.nl/~Hans.Leidekker/lexicon

GNA forum: http://admin.gnacademy.org:8001/uu-gna/text/moo/forum.htm

HumberHumbert's LambdaMOO Archive:
http://vesta.physics.ucla.edu/~smolin/lambda/

The Lost Library of MOOs: http://lucien.sims.berkeley.edu/moo.html

MOO-Cows FAQ Archive: http://www.moo.mud.org/moo-faq/

MOO Stats: http://lightsphere.com/moo/stats.html

MOO-WWW research directory: http://www.maths.tcd.ie/pub/mud/moo-www/rdir/rd.html

A MUD Workshop: http://klio.tema.liu.se/MUDworkshop

Various papers on MUDs and MOOs: http://sunsite.unc.edu/dbarberi/papers/