Habitat Object Manual

the Habitat object set described in excruciating detail

Chip Morningstar
Lucasfilm Ltd. Games Division
April 22, 1987

Introduction

This document describes the set of objects from which the Habitat fantasy worlds may be constructed. This document describes what the various objects are and specifies them in sufficient detail that they may be used in region design.

The Object Environment

Each object is a member of some *class* of objects, i.e., it is one of a group of objects that are all of the same basic type. An object's class determines what its behavior is going to be and how it is going to look on the screen. Any actual object is an *instance* of a particular class. This abstract model of classes and instances is used both in the C64 and in the host system.

The information describing a class consists of three kinds of *resources*: *imagery*, *sounds* and *behavior*. A class is described by a *class descriptor* which is just a data structure that points at these resources. The actual resources themselves are stored separately. This way, classes can share resources when they overlap between classes (which, in the case of behavior, is common).

Imagery consists of the animation cels or graphics driver tables required for the graphics software to display the object. Most objects consist of a single static image. A few have a small number of images, one of which is displayed at any given time depending on the state of the object. Some have a more complex set of images that are used to vary the appearance of the object according to some parameter (so that all objects of the class will not appear identical on the screen) or to animate the behavior of the object (Avatars are the primary example of this).

Sounds consist of the tables of data required to control the sound driver in creating various sound effects.

Behavior consists of a set of executable routines that are called by the object database software. These include the code to respond to any of the verbs used in the player interface. Other routines are specific to particular classes and will be called as the result of messages from the opposite end of the communications line (the host if this is the C64 system, and vice-versa).

An object's instance contains the object's `properties`. The properties are simply a set of values which describe that object as distinguished from other objects of the same class. There are certain properties which are possessed by all objects, regardless of class, while others are class-specific. The common properties are:

`obj_id` —
A global identifier in the object database (a 32-bit integer).

`style` —
Which of several possible variations of the class this is.

`x` —
The X-coordinate of the object's position in the region. If the object is inside a container, this value is ignored.

`y` —
The Y-coordinate of the object's position in the region. If the object is inside a container, this value

contains the position within the container that the object occupies. If the object is not in a container, the high-order bit (the 128's bit) encodes whether the object is in the background or the foreground. A 1 in this place indicates foreground, while a 0 indicates background. Background objects are not redrawn on the screen every frame, and so cannot be visually in front of any foreground objects, nor can they move around on the screen.

`orientation` —

This byte encodes a number of different pieces of information, as follows:

`0b00000001`	screen orientation: 1=face left, 0=face right
`0b00000010`	screen orientation: 1=forward, 0=back (Avatar only)
`0b00000100`	currently not used
`0b01111000`	choice of one of 16 patterns or Commodore 64 colors
`0b10000000`	decides between pattern (=0) or color (=1)

`gr_state` —

Is the graphic state of the object. Usually this is zero, but may vary depending on the class and style of the object. The interpretation of this for each class will be described with the classes themselves below.

`container` —

The global ID of the object, avatar or region that contains this object.

`restricted` —

A bit that flags whether this object is "restricted". Restricted object may not be carried through restricted region exits nor may they be placed in unrestricted containers. This provides a mechanism for preventing, for example, library books from being removed from the Library.

The Object Descriptions

What follows is a series of object descriptions, one for each class of objects in the basic object set. Each description has the following form:

Object:

name of object

Description:

A brief description of the object.

Function:

The object's purpose in the Habitat world.

Notes:

Anything else that needs to be said about the object.

Class: class number

Styles:

Stylistic variations possible for this object. In the format:

`0` — image style 0
- `0` — image for state #0
- `1` — image for state #1
- `2` — etc.

`1` — image style 1

`2` — and so on

Properties:

Properties that this object possesses. In the format:

`foobar` —
description of the foobar property

`bazbletch` —
description of the bazbletch property

`etc` —
and so on

Command Behavior:

Do:

The action to take for the verb **do**.

Go:

The action to take for the verb **go**.

Get:

The action to take for the verb **get**.

Put:

The action to take for the verb **put**.

Talk:

The action to take for the verb **talk**.

Reversed Do:

The action to take for the verb **reversed do**.

Asynchronous actions:

Other behavior that the object should be capable of as a result of asynchronous messages from the host or calls from other objects or other parts of the system.

C64 System Standard Actions

In the object descriptions that follow, these actions are often used for object behaviors. These standard actions are provided because the objects share a common underlying world model and there is much redundancy in the user interface. They provide both consistency and a significant reduction in the amount of code which must be created and stored.

`adjacentOpenCloseContainer`

If player is adjacent to container, try to open/close it. Otherwise, `depends`. Open fails if the container is locked and the Avatar isn't holding the key. If the Avatar is holding the key while closing it, the container will be locked.

`askOracle`

Send the player's text message to the Oracle, marked with the identity of the sender and the oracular object spoken to.

`broadcast`

Broadcast the typed text to everyone in region (as speech).

`depends`

Punt to the **reversed do** behavior of the object in the Avatar's hands, if there is one. If the Avatar is empty handed, `noEffect`.

`doMagic`

If holding the item, activate its (host defined) magical function, whatever that may be. If not holding it, `depends`.

`doMagicIfMagic`

Like `doMagic`, except that nothing happens if the item is not magical.

`flatGo`

If ground or wall, `goToCursor` (for wall the Y-coordinate will be clamped at the base of the object); for sky, `goToSky`; and for impassable, `noEffect`.

`getMass`

If the mass of the object is 0, try to walk to the object and pick it up. If the mass is 1, fail.

`goTo`

Walk to nearest position adjacent to the object's location, if the path is not obstructed.

`goToAndDropAt`

Walk to the indicated location and put the item in the Avatar's hand (if any) at that spot.

`goToAndDropInto`

If holding something in hand, walk to the designated item (which is a container) and try to put the object in hand into it.

`goToAndDropIntoIfOpen`

If holding something in hand and the designated object (which is a container) is open, walk to the container and try to put the object in hand into it.

`goToAndFill`

Walk to the designated object if you can; then, if the object in the avatar's hand is an empty container, fill it up with water.

`goToAndGet`

Walk to the object and pick it up, if possible.

`goToAndPickFrom`

Walk to the designated object (which is a container) and try to get something out of it. The object is guaranteed to be open.

`goToAndPickFromIfOpen`

Walk to the designated object (which is a container) and try to get something out of it. Fails if the object is not open.

`goToAndPickFromOrGet`

Walk to the designated item (which is a container). If it is open, try to get something out of it. If it is closed, try to pick it up.

`goToCursor`

Go to the location designated by the cursor.

`goToFurniture`

If adjacent to the indicated piece of furniture, sit in it. If already sitting in it, stand up. Otherwise, walk to it.

`goToSky`

Walk the Avatar to the region indicated by the object's `connection` property.

`illegal`

Does nothing, but should never happen.

`noEffect`

Do nothing.

`rdoMagic`

Activate the item's (host defined) magical function, whatever that may be, and direct the magic at whatever the player is pointing at.

`rdoMagicIfMagic`

Like `rdoMagic`, except that nothing happens if the item is not magical.

`shoot`

Attack the designated item or avatar with a firearm. Does damage to the person or thing appropriate to the sort of gun it is.

`strike`

Attack somebody or something with a close-range melee weapon. Does damage to the person or thing appropriate to the sort of weapon it is. Player must be standing next to the target in order to strike it.

`throw`

Throw the item in hand to the location the player is pointing at.

`ORACLESPEAK`

Utter asynchronous text message from the host when an oracle says something.

The Objects

The object descriptions follow on the proceeding pages:

Object:

amulet

Description:

A hang-around-the neck type amulet.

Function:

Generic magic talisman.

Notes:

Amulets are always magical. An amulet will do something special and powerful. The magical function may vary. Amulets are particularly rare and contain unusually powerful forms of magic. For the time being, amulets are merely artifacts, i.e., Avatars can't wear them, just hold them. They are magical without being worn.

Class: `2`

Styles:

`0` — amulet

Properties:

`magic_type` —

Sort of magic this amulet contains.

`magic_data` —

A 32-bit number interpreted by the magic routines. It's meaning varies depending on `magic_type`.

Command Behavior:

Do:

`doMagic`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`rdoMagic`

Asynchronous actions:

None.

Object:

aquarium

Description:

Your basic fish tank.

Function:

Household decoration.

Notes:

The aquarium is just a visual joke.

Class: `129`

Styles:

`0` — aquarium

- `0` — empty fish tank
- `1` — fish swimming in tank
- `2` — fish swimming without tank

Properties:

None

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`throw`

Object:

atm

Description:

Automatic **T**oken **M**achine.

Function:

Dispenses tokens.

Notes:

Every player in the Habitat has a bank account. Money that is in your bank account cannot be taken from you by force (i.e., it doesn't protect you from blackmail, fraud, etc.). The ATM is a device for accessing the account.

Class: `4`

Styles:

`0` — atm
- `0` — free standing
- `1` — wall mounted

Properties:

None.

Command Behavior:

Do:

Causes the machine to tell you your account balance.

Go:

`goTo`

Get:

Prompts you for an amount, then withdraws that many tokens from your bank account and puts it in your Avatar's hand as a Token object. If you ask for more money than you have available, it gives you what it can (i.e., your entire remaining balance).

Put:

If you are holding any tokens, they are deposited into your bank account. The machine then tells you your new balance.

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

avatar

Description:

You or somebody else.

Function:

The animated figure.

Notes:

The avatar object is one of the keystones of the system. Everything that happens is oriented around one or another avatars performing some action. As a result, the state information associated with an avatar is quite complex.

Class: `1`

Styles:

`0` — ordinary avatar

`17` —
penquin

`18` —
spider

`19` —
dragon

Properties:

`health` —
A number from 0 to 255 that indicates how healthy the Avatar is. 255 is the peak of health while 0 means the Avatar is dead.

`bank_account_balance` —
How many tokens are in the player's bank account.

`turf` —
The region number of the Avatar's Turf.

`stun_count` —
The Avatar's current stun count. Usually this will be 0 indicating that the Avatar is not stunned at all. If stunned, the count will be a low number such as 2 or 3.

`customize(3)` —
Three bytes containing the Avatar's current customization information. The low nybble of `customize(1)` is the pattern of the Avatar's arms, while the high nybble is the pattern of the Avatar's torso. The low nybble of `customize(2)` contains the pattern of the Avatars's legs. The high nybble is not used yet, nor is any of `customize(3)`.

`curse_immunity` —
This bit is set to indicate that the Avatar is immune to curses for the rest of the day.

`true_orientation` —
A "permanent" value for the Avatar's `orientation` property, so that we can change the latter and then restore it.

`true_head_style` —
A "permanent" value for the style of the Avatar's head, so that we can change it and then put it back.

`true_custom(3)` —
"Permanent" values for the Avatar's `customize(3)` properties so we can change them and then restore them.

`curse_type` —
The Avatar's current curse, if any. If the Avatar is not cursed (which will usually be the case), this is 0.

`curse_counter` —
A counter which is interpreted based on `curse_type`.

Command Behavior:

Do:

If the Avatar indicated is you, execute the **do** behavior associated with the object in hand, if any (this makes pointing simpler). If the Avatar is somebody else, "tag" them if you are empty handed or punt to `depends` if you are not.

Go:

If the Avatar indicated is you, change your posture (currently cycles between sitting and standing positions). If it is someone else, `goTo` them.

Get:

If the Avatar indicated is you, pick an item out of your pockets (fails if memory overflows). If it is someone else, try to grab whatever is in their hands (fails if they aren't holding anything). Fails in either case if you are already holding something.

Put:

If the Avatar indicated is you, try to put whatever you are holding into your pockets (fails if your pockets are full), unless it is a head, in which case you want to try to put it on (fails if you already have a head). If the Avatar is someone else, try to hand whatever you are holding to them (fails if they are already holding something). Fails in all cases if you are empty handed.

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

bag

Description:

The sack.

Function:

Can carry multiple items using only one hand.

Notes:

A bag is simply an object carrier that can be carried in the hands. It can hold up to 5 objects.

Class: `6`

Styles:

- `0` — bag
  - `0` — empty (closed)
  - `1` — full (open)
- `1` — suitcase
  - `0` — closed
  - `1` — open

Properties:

`open_flags` —

Flags showing whether the bag is open and/or unlocked. The 1's bit encodes open (1=open, 0=closed) while the 2's bit encodes unlocked (2=unlocked, 0=locked).

`key_hi` —

The high byte of the key number to lock/unlock the bag

`key_lo` —

The low byte of the key number to lock/unlock the bag

Command Behavior:

Do:

`adjacentOpenCloseContainer`

Go:

`goTo`

Get:

`goToAndPickFromOrGet`

Put:

`goToAndDropIntoIfOpen`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

ball

Description:

Your standard spherical throwing toy.

Function:

For playing games.

Notes:

The ball is relatively inert. You can pick it up and throw it.

Class: `7`

Styles:

`0` — ball

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

bed

Description:

An ordinary bed.

Function:

Can be sat on.

Notes:

The bed is a type of scenic object, usually found in building interiors. Up to two Avatars can sit on it. This doesn't really change anything, it's just for appearance.

Class: `130`

Styles:

`0` — bed (side view)
- `0` — with pillow
- `1` — without pillow

Properties:

`open_flags` —

Flags showing whether the bed is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:

`depends`

Go:

`goToFurniture`

Get:

`goToAndPickFrom`

Put:

`goToAndDropInto`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

book/newspaper

Description:

A readable document.

Function:

Displays text or artwork on paper.

Notes:

A book's contents are kept in the host. These are downloaded (in pieces) on demand when the player tries to read them. We use the full-screen text editor routines that are used for mail and pieces of paper, but we use them in a read-only mode that allows us to handle larger documents (since the contents may be discarded after they have scrolled off the screen).

Class: `10`

Styles:

- `0` — book end view
- `1` — book side view
- `2` — magazine
- `3` — newspaper
- `4` — sheet of paper
  - `0` — blank paper
  - `1` — paper with writing
  - `2` — envelope
- `5` — candelabra
- `6` — bottle
  - `0` — empty (laying on side)
  - `1` — full (standing up)
- `7` — stone tablets
- `8` — stock certificate

Properties:

`text_id` —
: Which document in the text database this book represents.

`current_page` —
: The page of this document currently being read (or most recently read).

`last_page` —
: The last page which may be read in this document.

Command Behavior:

Do:

If the player is holding the book, try to read it. Otherwise `depends`.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

boomerang

Description:

Your standard Australian throwing toy.

Function:

Joke.

Class: `11`

Styles:

`0` — boomerang

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Object:

bottle

Description:

A glass bottle.

Function:

Holds water, other liquids.

Notes:

The bottle can hold liquids. There is only one liquid substance defined right now, water, but we may add others as the design evolves (e.g., oil, fuel, etc.). The bottle is either filled or empty. There are no in-between states.

Class: `12`

Styles:

- `0` — bottle
  - `0` — empty (laying on side)
  - `1` — full (standing up)

Properties:

`filled` —

Flag telling whether bottle is filled (=1) or empty (=0).

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`Pour the contents of the bottle on the indicated spot. Of course, you can only do this with a full bottle.`

Asynchronous actions:

None.

Object:

box

Description:

Your basic box.

Function:

Can carry more than one item using only one set of hands.

Notes:

A box is simply an object carrier that can be carried in the hands. It is like a bag, but it has a larger capacity (10 instead of 5).

Class: `13`

Styles:

- `0` — box
  - `0` — closed
  - `1` — open
- `1` — treasure chest
  - `0` — closed
  - `1` — open

Properties:

`open_flags` —
Flags showing whether the box is open and/or unlocked. The 1's bit encodes open (1=open, 0=closed) while the 2's bit encodes unlocked (2=unlocked, 0=locked).

`key_hi` —
The high byte of the key number to lock/unlock the box

`key_lo` —
The low byte of the key number to lock/unlock the box

Command Behavior:

Do:
`adjacentOpenCloseContainer`

Go:
`goTo`

Get:
`goToAndPickFromOrGet`

Put:
`goToAndDropIntoIfOpen`

Talk:
`broadcast`

Reversed Do:
`throw`

Asynchronous actions:

None.

Object:

bridge

Description:

Your basic small foot or highway bridge.

Function:

Provides a pathway across water.

Notes:

This is a large scenic background object.

Class: `131`

Styles:

- `0` — bridge side view
  - `0` — entire side view
  - `1` — left end
  - `2` — center section
  - `3` — right end
  - `4` — left end with center section
- `1` — bridge end view
  - `0` — entire end view
  - `1` — left side
  - `2` — center section
  - `3` — right side
  - `4` — left side with center section

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goToCursor`

Get:
`noEffect`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

building

Description:

A house or building.

Function:

A background object to represent turfs and other structures.

Class: `132`

Styles:

`0` — house

- `0` — entire house, all details
- `1` — walkway only
- `2` — house body, no details
- `3` — entire house, no body windows
- `4` — body windows only
- `5` — doorway, no roof
- `6` — roof, no chimney
- `7` — roof, with chimney
- `8` — doorway, with roof
- `9` — chimney only
- `10` — house body with windows and chimney, no doorway

Properties:

`connection` —

The region to go to when **GO**ing to the building. If 0, the region adjacent to the current region in the direction of the building is used.

Command Behavior:

Do:

`depends`

Go:

`goToSky`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

bureaucrat-in-a-box

Description:

Your basic Habitat civil servant.

Function:

An interface between the players and the sysops.

Notes:

The bureaucrat-in-a-box is a special type of Oracle. It is a container than can hold one thing, which should be a head. This head is displayed as the bureaucrat's face.

Class: `158`

Styles:

`0` — jack-in-the-box bureaucrat

Properties:

`open_flags` —

Flags showing whether the bureaucrat is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`askOracle`

Reversed Do:

`illegal`

Asynchronous actions:

None

Object:

bush

Description:

Your basic bushy plant.

Function:

Scenic element. Obstruction.

Notes:

This is a fairly inert scenic element, provided almost entirely for visual appeal.

Class: `133`

Styles:

- `0` — round bush
  - `0` — with trunk
  - `1` — without trunk
- `1` — "flaming" bush
- `2` — spindly bush
  - `0` — without leaves
  - `1` — with leaves
  - `2` — with blinking leaves
- `3` — mushrooms
  - `0` — patch of 4 mushrooms
  - `1` — group of 3 mushrooms on left
  - `2` — single mushroom on right
- `4` — grass
  - `0` — tall and short
  - `1` — tall only
  - `2` — short only

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

chair

Description:

Your basic chair.

Function:

Can be sat in.

Notes:

The chair is a type of scenic object, usually found in building interiors. An Avatar can sit in it. This doesn't really change anything, it's just for appearance.

Class: `134`

Styles:

`0` — chair front view
`1` — chair side view
`2` — bar stool ("front view")
`3` — bar stool ("side view" — but really same as front view)

Properties:

`open_flags` —

Flags showing whether the chair is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:
`depends`
Go:
`goToFurniture`
Get:
`goToAndPickFrom`
Put:
`goToAndDropInto`
Talk:
`broadcast`
Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

change-o-matic

Description:

An amazing gadget.

Function:

Allows players to customize the color and pattern of items in their turves.

Class: `84`

Styles:

`0` — change-o-matic

- `0` — quiescent machine
- `1` — device operating

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

Change the color/pattern of the object pointed to to the next color/pattern in a pre-determined sequence. This only works inside your turf with immovable objects and with buildings immediately outside your turf.

Asynchronous actions:

None.

Object:

chest

Description:

A chest of drawers.

Function:

Can hold stuff.

Notes:

The chest of drawers is both a piece of furniture, for scenic purposes, and a container. It can hold up to 20 items.

Class: `135`

Styles:

- `0` — chest of drawers, front view
  - `0` — closed
  - `1` — open
- `1` — door (closet)
  - `0` — closed
  - `1` — open
- `2` — chest of drawers, side view

Properties:

- `0` — closed
- `1` — open

`open_flags` —

Flags showing whether the chest is open and/or unlocked. The 1's bit encodes open (1=open, 0=closed) while the 2's bit encodes unlocked (2=unlocked, 0=locked).

`key_hi` —

The high byte of the key number to lock/unlock the chest

`key_lo` —

The low byte of the key number to lock/unlock the chest

Command Behavior:

Do:

`adjacentOpenCloseContainer`

Go:

`goTo`

Get:

`goToAndPickFromIfOpen`

Put:

`goToAndDropIntoIfOpen`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

club

Description:

The most basic weapon.

Function:

Pain and injury at close range.

Notes:

The club is the simplest weapon. You go up to somebody and you hit them with it.

Class: `16`

Styles:

- `0` — club
  - `0` — horizontal
  - `1` — diagonal
  - `2` — vertical

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`strike`

Asynchronous actions:

None.

Object:

coke machine

Description:

Soda pop vending machine.

Function:

Decorative scenic object, joke.

Notes:

The coke machine mostly just sits there. If you put a Token in it it will take it, but it will never actually give you a coke.

Class: `136`

Styles:

`0` — coke machine

Properties:

`take` —

The cumulative income generated by this machine. This value should not be altered manually.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

Try to buy a coke (nothing will happen except the payment).

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

compass

Description:

Your basic pointer to the West Pole.

Function:

Tells absolute direction.

Notes:

The compass doesn't actually DO anything. It is simply displayed in a manner which indicates the direction that is West.

Class: `17`

Styles:

`0` — compass

- `0` — pointing up
- `1` — pointing right
- `2` — pointing down
- `3` — pointing left
- `4` — pointer spinning

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

couch

Description:

Your basic living room couch.

Function:

Like chair, but can be sat in by multiple avatars.

Notes:

A couch works just like a chair, but it can hold two avatars.

Class: `137`

Styles:

`0` — couch, front view

`1` — couch, side view (not supported)

Properties:

`open_flags` —

Flags showing whether the couch is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:

`depends`

Go:

`goToFurniture`

Get:

`goToAndPickFrom`

Put:

`goToAndDropInto`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

countertop

Description:

Store counter.

Function:

Can support things. Mediates transactions.

Notes:

The countertop is a mechanism for mediating transactions. Objects placed on the counter follow a slightly different protocol when being picked up by passers by. If you place an object on the counter, you can pick it up again. However, if somebody else places an object on the counter, you can only pick it up if you already placed an item of your own on the counter. This interlock is mediated by the host, when it decides whether a GET operation has succeeded or not. The countertop can contain up to 5 items.

Class: `18`

Styles:

`0` — countertop

Properties:

`open_flags` —

Flags showing whether the countertop is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

`whoput(5)` —

The global ID's of the Avatars who put things down on the counter. There is one entry for each slot in the container. These values ordinarily should not be altered manually.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndPickFrom` (subject to transaction constraints described above.)

Put:

`goToAndDropInto`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

crystal ball

Description:

Your basic crystal ball.

Function:

Oracle and diviner.

Notes:

The crystal ball functions rather like the fountain, but on a more personal level. It answers questions, makes predictions, and generally does all of the things that oracles are supposed to do. However, it is, by tradition, subtle and not completely reliable. Sometimes it answers your questions immediately. Sometimes it even carries on a conversation with you. Most of the time though, it takes quite a while to get an answer: you have to come back several days later. Occasionally it says things spontaneously. Such things are usually important. Anyone in the region with the crystal ball can hear what it says. Of course, the minds behind the ball are our own. Somebody has to respond, but the nature of the oracle business is such that a response need not be timely, nor need all questions be answered.

Class: `20`

Styles:

`0` — crystal ball
- `0` — plain
- `1` — sparkling

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
`askOracle`

Reversed Do:
`throw`

Asynchronous actions:

ORACLESPEAK

Object:

die

Description:

A six-sided die.

Function:

For gambling, games.

Notes:

The die is a simple object, like the compass, whose only real function is to display its own state.

Class: `21`

Styles:

- `0` — single die
  - `0` — rolling
  - `1-6` — 1 through 6
- `1` — pair of dice
  - `0` — single die rolling
  - `1-6` — single die 1 through 6
  - `7` — pair rolling
  - `8-43` — pairs 1-1 through 6-6

Properties:

None.

Command Behavior:

Do:

Roll the die (dice).

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

display case

Description:

Store display case.

Function:

Can hold things visibly but safely, even if unattended.

Notes:

The display case is a transaction mediator similar to the countertop. It has an owner. The protocol that it obeys is that objects placed in it by the owner may only be picked up again by owner. Thus, objects can be left open to view with impunity, even if unattended. The display case may hold up to 5 items.

Class: `22`

Styles:

`0` — display case
- `0` — entire case
- `1` — without legs
- `2` — without shelf
- `3` — without center shelf

Properties:

`open_flags` —
: Flags showing whether the display case is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

`owner` —
: The global ID of the Avatar who owns this display case (0 if unowned).

`locked(5)` —
: Bits that encode whether the corresponding slots in the contents list contain objects which may only be removed by the owner.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`goToAndPickFrom` (subject to the constraints of the above protocol.)

Put:
`goToAndDropInto`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

door

Description:

The conventional door.

Function:

Graphic element in buildings. Passageway through wall.

Notes:

A door is like a wall section, except that it can be opened to allow passage through it. You can only open a door if it is unlocked or if you have the key.

Class: `23`

Styles:

- `0` — plain door
  - `0` — closed
  - `1` — open
- `1` — door with window
  - `0` — closed
  - `1` — open

Properties:

`open_flags` —

Flags showing whether the door is open and/or unlocked. The 1's bit encodes open (1=open, 0=closed) while the 2's bit encodes unlocked (2=unlocked, 0=locked).

`key_hi` —

The high byte of the key number to lock/unlock the door

`key_lo` —

The low byte of the key number to lock/unlock the door

`connection` —

The region to go to when **GO**ing through the door. If 0, the region adjacent to the current region in the direction of the door is used.

Command Behavior:

Do:

If adjacent, open/close the door. Otherwise, depends. Open fails if the door is locked and the Avatar isn't holding the key. If the Avatar is holding the key while closing it, the door will be locked.

Go:

If the door is closed or the player is pointing at the frame or the door itself (as opposed to the empty opening), walk to the door. Otherwise, walk through the door to the region it is connected to.

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

dropbox

Description:

A convenient roadside mail drop box.

Function:

Obsolete interface to mail system.

Notes:

This used to be an object for mailing letters. Now it is just scenic.

Class: `24`

Styles:

`0` — dropbox

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

drugs

Description:

Little pills.

Function:

A magic-like hook for arbitrary effects.

Notes:

A drug object is represented as a pill bottle. The bottle contains a certain number of pills. This number is decremented each time you take one. When they run out the bottle is no longer effective. Taking a pill causes something to happen. There are a variety of possible effects.

Class: `25`

Styles:

`0` — pill bottle

Properties:

`count` —

How many pills are left.

`effect` —

The effect of the pills. This number selects one of several different possible effects.

Command Behavior:

Do:

Take a pill and apply it's (host defined) effect, if any. Decrement the pill count.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

elevator

Description:

It takes you up and down.

Function:

A form of limited-range teleport that doesn't require money to use.

Notes:

An elevator operates like a teleport, but you don't activate it with a token. Instead, it is just sort of "permanently active". In addition, elevators use a special two-part area code (the first part of which is invisible to the players) so that you can't jump from elevators to teleports (i.e., cadge free teleport rides off of the system) but only go from floor to floor within a building.

Class: `28`

Styles:

`0` — jump tube
- `0` — entire elevator
- `1` — without control box
- `2` — without ground skirt
- `3` — without control box or ground skirt
- `4` — without ground skirt or rivets
- `5` — without control box, ground skirt, or rivets

Properties:

`address` —

The "teleport address" of the elevator.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

"Teleport" the player to the address that is the talk string typed by the player.

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

escape device

Description:

Your basic panic button.

Function:

Gets you out of a jam, fast.

Notes:

This device is very rare and expensive. It is a little box with a button on it. If you press the button, you are instantly teleported back to your turf. Since this is such a powerful device, it is given only a limited charge. Each teleport uses one unit of charge. When the charge level reaches zero, the device stops working. Note that the typical charge found in practice should be 1.

Class: `26`

Styles:

`0` — escape device

Properties:

`charge` —

How many teleports are left in this device before it runs out.

Command Behavior:

Do:

If the player is holding the device, press the button and bug out. If not holding, `depends`.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

fake gun

Description:

A pistol (almost).

Function:

Apparant death and destruction from a distance, but not really.

Notes:

The fake gun works like the real gun, except when you shoot with it a flag that says "BANG!" comes out instead of actually shooting somebody.

Class: `27`

Styles:

`0` — fake gun
- `0` — plain
- `1` — with “BANG” flag

Properties:

None

Command Behavior:

Do:

If the player is holding the gun and the flag is out, reset it.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

Act as if shooting the gun, then have the flag pop out.

Asynchronous actions:

None.

Object:

fence

Description:

A section of an impassable, man-made barrier.

Function:

Linear obstruction.

Notes:

Each fence object is a single section of fence, running horizontally with respect to the viewpoint. Fence sections are of a standard size. A fence blocks passage across the line that it runs on. For visual stylistic reasons, there are several different styles of fencing available.

Class: `138`

Styles:

`0` — chain-link fence
- `0` — left post and fencing
- `1` — post only
- `2` — fencing only

Properties:

None

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

flag

Description:

The colors.

Function:

Scenic decoration.

Notes:

A flag is a decoration as well as a marker that can be carried around.

Class: `29`

Styles:

`0` — flag
- `0` — phase A
- `1` — phase B
- `2` — phase C

Properties:

`mass` —

A flag that indicates whether or not the object can be picked up.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`getMass`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`throw`

Asynchronous actions:

None

Object:

flashlight

Description:

Your basic hand torch.

Function:

Portable light source at night.

Notes:

The Habitat can follow a cycle of day and night. When it is night-time, the screen is displayed in darkened colors, which makes things hard to see. Having a flashlight in the region makes the region appear like daytime (provided that the light is turned on, of course), even if it is night elsewhere. We may wish to consider requiring batteries, but for now it just works forever.

Class: `30`

Styles:

- `0` — flashlight
  - `0` — off
  - `1` — on (actually, same as off)
- `1` — torch
  - `0` — unlit
  - `1` — burning
- `2` — candelabra
  - `0` — off
  - `1` — on (actually, same as off)

Properties:

`on` —

Flag telling whether the light is on (=1) or off (=0).

Command Behavior:

Do:

If the player is holding the flashlight, turn it on or off. Otherwise, `depends`.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

flat

Description:

Your basic background.

Function:

Provide the backdrop for regions.

Notes:

A flat is a multi-purpose background object. It comes in several different size variants, however it is basically just a colored rectangle. There are four different types of flat. A flat's type governs how it is treated for purposes of walking. The four types correspond to the notions of ground, sky, wall and impassable ground.

Class: 93

Styles:

- 0 — tall ground rectangle
- 1 — short ground rectangle
- 2 — "mod" pattern wall
- 3 — short wall rectangle
- 4 — plain wall
- 5 — brick wall
- 6 — "mod" sky
- 7 — medium sky rectangle
- 8 — mountain
  - 0 — with snowcap
  - 1 — without snowcap
  - 2 — snowcap only
- 9 — cave mouth
  - 0 — rounded top
  - 1 — square top
- 10 — ball
- 11 — tree/cloud
  - 0 — entire cloud
  - 1 — upper right corner
  - 2 — lower left corner
  - 3 — right half
  - 4 — bottom half
- 12 — pipes
  - 0 — full regalia
  - 1 — horizontal pipe section
  - 2 — horizontal section with fitting
  - 3 — horizontal section with upward elbow on left
  - 4 — L-shaped secton
  - 5 — vertical section with elbow at bottom
  - 6 — vertical section with fitting
  - 7 — horizontal fitting only
  - 8 — vertical fitting with small section
  - 9 — vertical elbow only
  - 10 — vertical section with elbow
  - 11 — vertical fitting only
  - 12 — horizontal section with corner on right
  - 13 — valve

`14` — vertical section with fitting
`15` — right corner
`16` — short vertical section
`17` — vertical elbow

Properties:

`flat_type` —
What sort of flat this is: 0=sky, 1=wall, 2=ground, 3=impassable.

Command Behavior:

Do:
`depends`

Go:
`flatGo`

Get:
`noEffect`

Put:
`goToAndDropAt`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

floor lamp

Description:

A household floor lamp.

Function:

Provides light at night.

Notes:

The Habitat can follow a cycle of day and night. When it is night-time, the screen is displayed in darkened colors, which makes things hard to see. Having a floor lamp in the region makes the region appear like daytime (provided that the lamp is turned on, of course), even if it is night elsewhere. The floor lamp is a decorative furniture object also.

Class: `139`

Styles:

- `0` — floor lamp
  - `0` — off, with shade
  - `1` — on, with shade
  - `2` — off, without shade
  - `3` — on, without shade
  - `4` — off, with shade and bent neck
  - `5` — on, with shade and bent neck
  - `6` — off, with bent neck, no shade
  - `7` — on, with bent neck, no shade
  - `8` — off, bent neck and shade, no base
  - `9` — on, bent neck and shade, no base
  - `10` — off, bent neck, no shade, no base
  - `11` — on, bent neck, no shade, no base

Properties:

`on` —

Flag telling whether or not the lamp is turned on (1=on).

Command Behavior:

Do:

Walk to the lamp if not already adjacent to it, and turn it on or off.

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

fortune machine

Description:

A vending machine that tells your fortune.

Function:

Humor.

Notes:

The fortune machine is a little like the oracle, but less profound. You give it money and it tells you a fortune. The fortunes are little proverbs, sayings, predictions and pieces of advice in the tradition of fortune cookie programs everywhere.

Class: `140`

Styles:

`0` — fortune machine

- `0` — globe, box, pedestal and base
- `1` — globe, pedestal and base
- `2` — parking meter with pedestal and base
- `3` — globe only
- `4` — globe and pedestal
- `5` — globe and box

Properties:

`take` —

The cumulative income generated by this machine. This value should not be altered manually.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

If holding tokens, put money into the machine and buy a fortune. The price of the fortune will be deducted from the amount in hand and the fortune itself will appear in a balloon message.

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

fountain

Description:

Generic looking tacky town square fountain.

Function:

Scenic element. Water source. Oracle.

Notes:

The fountain is a major scenic element, but its most dramatic function is as oracle. The oracle answers questions, makes predictions, and generally does all of the things that oracles are supposed to do. However, it is, by tradition, subtle and not completely reliable. Sometimes it answers your questions immediately. Sometimes it even carries on a conversation with you. Most of the time though, it takes quite a while to get an answer: you have to come back several days later. Occasionally it says things spontaneously. Such things are usually important. Anyone in the region with the oracle can hear what it says. Of course, the minds behind the oracle are our own. Somebody has to respond, but the nature of the oracle business is such that a response need not be timely, nor need all questions be answered.

Class: `141`

Styles:

`0` — fountain
- `0` — with cherub and spout
- `1` — spout, no cherub
- `2` — no spout, no cherub
- `3` — cherub only

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`goToAndFill`

Put:
`noEffect`

Talk:
`askOracle`

Reversed Do:
`illegal`

Asynchronous actions:

ORACLESPEAK

Object:

frisbee

Description:

Your standard disk throwing toy.

Function:

For playing games.

Notes:

The frisbee is relatively inert. You can pick it up and throw it. For the time being, frisbee operates just like the ball.

Class: `31`

Styles:

- `0` — frisbee
  - `0` — edge view
  - `1` — top view

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

game piece

Description:

A typical board game marker.

Function:

Enables board games such as checkers, chess and backgammon.

Notes:

The game piece is a simple marker that can be moved around.

Class: 5

Styles:

0 — game piece
- 0 — pawn
- 1 — bishop
- 2 — knight
- 3 — rook
- 4 — king
- 5 — queen
- 6 — checker
- 7 — checker king
- 8 — backgammon piece
- 9 — backgammon piece

Properties:

None.

Command Behavior:

Do:

Change the game piece's state depending on its current state and what style of piece it is. Thus checker pieces are turned into kings while reversi pieces simply change color.

Go:

`goTo`

Get:

Pick the object up telekinetically.

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

garbage can

Description:

Conventional garbage can or wastebasket.

Function:

Makes things disappear.

Notes:

The garbage can is a means for getting rid of things. It operates like any other container, except that upon command it will cause to disappear anything inside it. The garbage can will hold up to 20 items.

Class: `32`

Styles:

`0` — garbage can
- `0` — closed
- `1` — open

`1` — kitchen wastebasket
- `0` — closed
- `1` — open

Properties:

`open_flags` —
Flags showing whether the garbage can is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:
Flush the garbage can, causing any objects inside it to disappear.

Go:
`goTo`

Get:
`goToAndPickFrom`

Put:
`goToAndDropInto`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None

Object:

gemstone

Description:

Like a rock, only worth more.

Function:

Valuable. May contain magic.

Notes:

Some objects in the Habitat are magical. A magical object is one that does something special and powerful in addition to whatever other things the object would do intrinsically. The magical function may vary.

Class: `33`

Styles:

`0` — gemstone
- `0` — plain
- `1` — sparkling

Properties:

`magic_type` —
Sort of magic this gemstone contains. 0 if not magical.

`magic_data` —
A 32-bit number interpreted by the magic routines. It's meaning varies depending on `magic_type`.

Command Behavior:

Do:
`doMagicIfMagic`

Go:
`goTo`

Get:
`goToAndGet`

Put:
`goToAndDropAt`

Talk:
`broadcast`

Reversed Do:
`rdoMagicIfMagic`

Asynchronous actions:

None.

Object:

ghost

Description:

Invisible Avatars who act like they're not there.

Function:

Avoids traffic jams and allows theaters to operate.

Notes:

The ghost object simply holds the graphic image for the icon that appears when there are ghost(s) in the region with you.

Class: `3`

Styles:

`0` — ghost icon

Properties:

None.

Command Behavior:

Do:

`noEffect`

Go:

`noEffect`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

glue

Description:

Invisible substance.

Function:

Holds objects together as if they were a single object.

Notes:

Glue is used to fasten down non-scenic object in order to use them as part of the scenery. The user interface should never allow players to point to glue. The glue may hold up to six items, each with an individual (x,y) offset position from the glue's position.

Class: `98`

Styles:

`0` — invisible glue

Properties:

`open_flags` —
Flags showing whether the glue is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

`x_offset_1` —
The X-offset from the glue's position of the first item in the glue's contents list.

`y_offset_1` —
The Y-offset from the glue's position of the first item in the glue's contents list.

`x_offset_2` —
The X-offset from the glue's position of the second item in the glue's contents list.

`y_offset_2` —
The Y-offset from the glue's position of the second item in the glue's contents list.

`x_offset_3` —
The X-offset from the glue's position of the third item in the glue's contents list.

`y_offset_3` —
The Y-offset from the glue's position of the third item in the glue's contents list.

`x_offset_4` —
The X-offset from the glue's position of the fourth item in the glue's contents list.

`y_offset_4` —
The Y-offset from the glue's position of the fourth item in the glue's contents list.

`x_offset_5` —
The X-offset from the glue's position of the fifth item in the glue's contents list.

`y_offset_5` —
The Y-offset from the glue's position of the fifth item in the glue's contents list.

`x_offset_6` —
The X-offset from the glue's position of the sixth item in the glue's contents list.

`y_offset_6` —
The Y-offset from the glue's position of the sixth item in the glue's contents list.

Command Behavior:

Do:
`illegal`

Go:
`illegal`

Get:
`illegal`

Put:
`illegal`

Talk:
`illegal`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

grenade

Description:

A little bomb that goes boom.

Function:

Death and mayhem in quantity.

Notes:

A grenade has a pin. It is harmless unless the pin is pulled. If the pin is pulled, then it will explode the next time it leaves the avatar's hand, blowing up everything at or near the destination location. Also, once the pin is pulled it may not be reinserted.

Class: `35`

Styles:

`0` — grenade

Properties:

`pinpulled` —

Flag that the pin has been pulled.

Command Behavior:

Do:

If the player is holding the grenade, pull the pin.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

Explodes after awhile, damaging nearby objects and avatars as if they'd been shot.

Object:

ground

Description:

The basic background below the horizon.

Function:

Can be walked on. Can be pointed at, returning a location.

Class: `36`

Styles:

`0` — tall ground rectangle
`1` — short ground rectangle

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goToCursor`

Get:
`noEffect`

Put:
`goToAndDropAt`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

gun

Description:

A pistol.

Function:

Death and destruction from a distance.

Notes:

A gun works pretty much as you would expect. You shoot by pointing at the target and selecting **do**. We should decide if requiring ammunition would be a good idea.

Class: `37`

Styles:

`0` —	gun type #1
`1` —	gun type #2
`2` —	gun type #3
`3` —	mauser

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`shoot`

Asynchronous actions:

None.

Object:

hand of god

Description:

The finger of doom.

Function:

A probably impractical system management tool.

Notes:

The hand of god is a giant animated hand that comes down off the top of the screen and fires a lightning bolt off its finger at a target, resulting a large explosion that leaves a small pile of smoking cinders in its wake. However, due to the size of the image we will probably never make it work.

Class: `38`

Styles:

`0` — hand of god with animating lightning bolt

`1` — pile of cinders

Properties:

None.

Command Behavior:

None: it is never selectable on the screen.

Asynchronous actions:

None.

Object:

head

Description:

Your basic head.

Function:

Decorative. Helps distinguish one avatar from another.

Notes:

A head is simply an object that serves to personalize an avatar. It is represented by a cel that simply follows the avatar's neck around.

Class: `127`

Styles:

`0` — There are numerous head styles. See the appendix.

Properties:

None.

Command Behavior:

Do:

If wearing the head, punt to the **do** operation for the avatar itself; otherwise `depends`,

Go:

`goTo`

Get:

If wearing the head, take it off. Otherwise, try to pick it up. Fails if the Avatar is not empty-handed or if the head is being worn by somebody else.

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

hole

Description:

A hole in the ground.

Function:

Allows us to hide things in plain sight.

Notes:

A hole is simply a container that is invisible when it is closed.

Class: `88`

Styles:

`0` — hole

- `0` — closed (invisible)
- `1` — open

Properties:

`open_flags` —

Flags showing whether the hole is open and/or unlocked. The 1's bit encodes open (1=open, 0=closed) while the 2's bit encodes unlocked (2=unlocked — the hole should be permanently unlocked).

`key_hi` —

The high byte of the key number to lock/unlock the hole (should be permanently set to 0).

`key_lo` —

The low byte of the key number to lock/unlock the hole (should be permanently set to 0).

Command Behavior:

Do:

Opens/closes the hole, but only if you are adjacent to it and holding a shovel.

Go:

`goTo`

Get:

`goToAndPickFromIfOpen`

Put:

`goToAndDropIntoIfOpen`

Talk:

`broadcast`

Reversed Do:

`noEffect`

Asynchronous actions:

None.

Object:

hot tub

Description:

A touch of Marin.

Function:

A place to hang out.

Notes:

The hot tub is a silly in-joke. In spite of its esoteric appearance, it actually operates more or less like ordinary ground. The only difference is that it has a front part and a back part so that the avatars can be sandwiched in between and thus look like they are in the water. It's also a water source.

Class: `144`

Styles:

`0` — hot tub
- `0` — entire tub
- `1` — front half

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goToCursor`

Get:
`goToAndFill`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

house cat

Description:

A lazy cat that lays around the house.

Function:

Scenic element. Puzzle with no solution.

Notes:

The housecat just lays there and sleeps. Every evening it moves to a different spot. You can't pick it up or move it.

Class: `143`

Styles:

`0` — house cat

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

instant object pill

Description:

A little pill, until you use it.

Function:

Add water, it turns into some object.

Notes:

A instant object pill looks like an ordinary pill, until you pour water on it, at which point it transforms into some other sort of object. A instant object pill can, conceivably, turn into anything.

Class: `40`

Styles:

`0` — instant object pill

Properties:

`instant_what` —

Class of object that this turns into.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

jukebox

Description:

The all-American Rockola.

Function:

Plays music.

Notes:

The jukebox is supposed to be a coin operated music machine. Currently it's just an inert scenic object. Someday this may change.

Class: `145`

Styles:

`0` — jukebox

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

key

Description:

A key for our unpickable locks.

Function:

Opens locked doors, containers.

Notes:

Each key object has a key number, as does each object with a lock (doors, containers, and so on). You can open locks with a key whose number matches that of the lock. It is possible for there to be more than one key object in the world with given key number, so that a lock can have multiple keys. To use a key, all you have to do is have it in your hand when you try to open the locked thing.

Class: `42`

Styles:

- `0` — ordinary key
  - `0` — type #1
  - `1` — type #2
- `1` — card key

Properties:

`key_number_hi` —
The high byte of the key number of this key.

`key_number_lo` —
The low byte of the key number of this key.

Command Behavior:

Do:
Displays the key number of the key.

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`throw`

Asynchronous actions:

None.

Object:

knick knack

Description:

Your basic gewgaw.

Function:

Scenic element. Possibly magical.

Notes:

Knick knacks are usually inert scenic objects, intended only to clutter up a scene for purposes of visual variety. Occasionally, a knick-knack will be magical. The set of possible gewgaws is nearly infinite. We will pick a reasonable sample of several different styles.

Class: `43`

Styles:

- `0` — candelabra
- `1` — trophy
- `2` — knick-knack
- `3` — vase of flowers
  - `0` — vase only
  - `1` — vase with flowers
  - `2` — flowers only
  - `3` — vase with flowers
- `4` — juggling balls
  - `0` — in a pile
  - `1` — juggling
- `5` — chainsaw
  - `0` — off
  - `1` — on
- `6` — teddy bear
- `7` — rubber ducky
- `8` — answering machine
  - `0` — light off
  - `1` — light on
  - `2` — light blinking
- `9` — telephone
- `10` — towel
- `11` — microphone
- `12` — road pizza
- `13` — cups
  - `0` — goblet
  - `1` — mug
  - `2` — wine glass
  - `3` — hi-ball glass
  - `4` — fancy drink glass

Properties:

`magic_type` —
Sort of magic this knick-knack contains. 0 if not magical.

`magic_data` —
A 32-bit number interpreted by the magic routines. It's meaning varies depending on `magic_type`.

Command Behavior:

Do:
`doMagicIfMagic`
Go:
`goTo`
Get:
`goToAndGet`
Put:
`noEffect`
Talk:
`broadcast`
Reversed Do:
`rdoMagicIfMagic`

Asynchronous actions:
None.

Object:

knife

Description:

Sharp pointy thing.

Function:

Death and injury at close range.

Notes:

The knife works like a club: you go up to somebody and attack them with it.

Class: `44`

Styles:

`0` — knife type #1
`1` — knife type #2

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`strike`

Asynchronous actions:

None.

Object:

magic lamp

Description:

Just like Aladdin had...

Function:

Rub it and a genie appears to grant a wish.

Notes:

The magic lamp object represents both the magic lamp and the genie. You rub on the lamp and the genie appears. The first thing you say to it is interpreted as your wish and then the genie disappears. The genie operates like the oracle: it takes down your request; at some point one of our system people reviews the request and does something in response. It may take a while for your wish to be answered, and even then the answer may not be what you want or expect. The genie only grants one wish per person. Once the genie has noted your wish he disappears, and the lamp disappears with him. If you need to make another wish you must find another lamp (and another genie).

Class: `45`

Styles:

`0` — magic lamp
- `0` — plain lamp
- `1` — genie appearing (and staying)
- `2` — genie talking
- `3` — genie disappearing

Properties:

`lamp_state` —
: What state the lamp is in, internally. Should be initialized to zero and then not altered manually.

`wisher` —
: The global ID of the Avatar making a wish, once the wishing starts.

Command Behavior:

Do:
If you are the one holding the lamp, rub it and make the genie appear. Otherwise, `depends`.

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
If the speaker is the person making a wish, interpret the message as the wish; otherwise, `broadcast`.

Reversed Do:
`throw`

Asynchronous actions:

After a certain amount of time, if a player has rubbed the lamp but has not made a wish, the Genie gives up and leaves, causing the lamp to disappear.

Object:

magic staff

Description:

A stick about as tall as an avatar.

Function:

Generic magic talisman.

Notes:

Staves are usually magical. An staff will do something special and powerful. The magical function may vary. Staves are relatively rare and contain unusually powerful forms of magic.

Class: `46`

Styles:

`0` — magic staff

Properties:

`magic_type` —

Sort of magic this staff contains.

`magic_data` —

A 32-bit number interpreted by the magic routines. It's meaning varies depending on `magic_type`.

Command Behavior:

Do:

`doMagic`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`rdoMagic`

Asynchronous actions:

None.

Object:

magic wand

Description:

Just like your fairy godmother has.

Function:

Generic magic talisman.

Notes:

Wands are usually magical. A wand will do something special but it not as powerful as a staff. The magical function may vary. Wands are relatively common as magical objects go (which is not very common at all) and usually contain fairly ordinary forms of magic.

Class: `47`

Styles:

`0` — high-tech wand
`1` — traditional magician's wand
`2` — teddy bear

Properties:

`magic_type` —
Sort of magic this wand contains.

`magic_data` —
A 32-bit number interpreted by the magic routines. It's meaning varies depending on `magic_type`.

Command Behavior:

Do:
`doMagic`

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`rdoMagic`

Asynchronous actions:

None.

Object:

mailbox

Description:

A conventional household roadside mailbox.

Function:

Obsolete interface to mail system.

Notes:

This used to be an object for sending and receiving mail. Now it is just scenic.

Class: `48`

Styles:

`0` — mailbox
- `0` — flag down
- `1` — flag up

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`noEffect`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

matchbook

Description:

An empty book of matches

Function:

Has an ad that you can read.

Notes:

The matchbook is actually an information carrying device. There are never any matches.

Class: `49`

Styles:

`0` — matchbook

Properties:

`text` —

Up to 114 characters of text message.

Command Behavior:

Do:

Read the text on the matchbook.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

movie/television camera

Description:

The basic tool of journalism and drama.

Function:

Provides a way to record ongoing events.

Notes:

Eventually, we would like this to be a way of recording the events in a region. For the time being it is just a prop.

Class: `52`

Styles:

- `0` — movie camera
  - `0` — camera only
  - `1` — camera with tripod
  - `2` — tripod only

Properties:

None.

Command Behavior:

Do:

Switch it between tripod mode and hand-held mode.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

paper

Description:

A piece of paper.

Function:

Can be written upon and then retrieved.

Notes:

Any piece of paper whose text begins "to: *name*" can be sent as a mail message.

Class: `54`

Styles:

`0` — piece of paper

Properties:

`text_id` —

Which document in the text database this paper represents.

Command Behavior:

Do:

If holding the piece of paper, open it up and read its contents (and allow the player possibly to write on it); otherwise, `depends`.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

pawn machine

Description:

A strange device found in pawnshops.

Function:

Allows players to recycle objects, converting them into tokens.

Notes:

The pawn machine operates like a box that can contain at most one item. However, an item placed in it may be converted into cash.

Class: `96`

Styles:

`0` — pawn machine

- `0` — quiescent
- `1` — operating

Properties:

`open_flags` —

Flags showing whether the machine is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:

If the player is adjacent to the machine and there is something in it, cause the contents of the machine to disappear and then dispense a token of the value of the object that was in the machine; otherwise, `noEffect`.

Go:

`goTo`

Get:

`goToAndPickFrom`

Put:

`goToAndDropInto`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

picture

Description:

A picture to display.

Function:

Displays artwork or text on wall or in space.

Notes:

A picture displays one of several static images stored on the object disk.

Class: `152`

Styles:

- `0` — variable sized picture frame
  - `0` — small bordered rectangle
  - `1` — large rectangle, borders on left and top
  - `2` — large rectangle, no borders
  - `3` — small rectangle, borders on right and bottom
  - `4` — large rectangle, borders on left and bottom
  - `5` — large rectangle, borders on left, top and bottom
- `1` — exit sign
- `2` — Island Vacation poster
- `3` — "Turf Sweet Turf" sign

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

plant

Description:

Your basic generic plant.

Function:

Scenic element. Obstruction.

Notes:

This is a fairly inert scenic element, provided almost entirely for visual appeal.

Class: `58`

Styles:

`0` — plant type #1
`1` — plant type #2
`2` — plant type #3

Properties:

`mass` —

A flag that indicates whether or not the plant can be picked up.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`getMass`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`throw`

Asynchronous actions:

None.

Object:

plaque

Description:

An informative plaque

Function:

Provides detailed information where the text required would not fit on a sign.

Notes:

This is a fairly inert scenic element, provided almost entirely for visual appeal.

Class: `55`

Styles:

- `0` — television monitor
- `1` — window
  - `0` — whole window
  - `1` — without sash
  - `2` — without left side
  - `3` — without left side, sash
  - `4` — without left side, bottom
  - `5` — without left side, sash, bottom
  - `6` — without right side, bottom
  - `7` — without right side, sash, bottom
- `2` — plaque

Properties:

`text_id` —

Which document in the text database this plaque represents.

`current_page` —

The page of this document currently being read (or most recently read).

`last_page` —

The last page which may be read in this document.

Command Behavior:

Do:

Read the text off the plaque (going into the book/paper interface to do so).

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

pond

Description:

A small (within the region) body of water.

Function:

Water source. Water obstruction.

Class: `147`

Styles:

`0` — pond

- `0` — entire pond
- `1` — water only
- `2` — water and stone
- `3` — water and grass
- `4` — stone only
- `5` — grass only

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndFill`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

region

Description:

A place in the world.

Function:

The basic building block of Habitat topography.

Notes:

The region is not really an object, but we have an object that represents it so that the host can send messages to it. Basically, all it does is contain things. It also has a few other properties that are used for background graphics rendering. Note that some of the region's generic properties are interpreted slightly differently. In particular, the `y` position encodes the vertical screen position of the horizon line.

Class: `0`

Styles:

None.

Properties:

None.

Command Behavior:

None: it is not a selectable object on the screen.

Asynchronous actions:

None.

Object:

ring

Description:

Your basic magic ring.

Function:

Generic magic talisman.

Notes:

Rings are always magical. A ring will do something special and powerful. The magical function may vary. Rings, like amulets, cannot be worn, though the magic they contain may be used.

Class: `60`

Styles:

`0` — ring
- `0` — plain
- `1` — sparkling

Properties:

`magic_type` —
Sort of magic this ring contains.

`magic_data` —
A 32-bit number interpreted by the magic routines. It's meaning varies depending on `magic_type`.

Command Behavior:

Do:
`doMagic`

Go:
`goTo`

Get:
`goToAndGet`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`rdoMagic`

Asynchronous actions:

None.

Object:

river

Description:

Body of water flowing through a region.

Function:

Water source. Linear water obstruction.

Class: `148`

Styles:

`0` — missing image

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndFill`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

rock

Description:

Your basic rock.

Function:

Scenic element. If small can be picked up, thrown.

Class: `61`

Styles:

`0` — small rock
`1` — medium rock
`2` — large rock

Properties:

`mass` —

A flag that indicates whether or not the rock can be picked up.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`getMass`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`throw`

Asynchronous actions:

None.

Object:

safe

Description:

Your basic office or household safe.

Function:

Stores things under lock and key.

Notes:

The safe is both a piece of furniture, for scenic purposes, and a container. It has two states, open and closed, and it can be locked. The safe can hold up to 10 items.

Class: `150`

Styles:

`0` — safe
- `0` — closed
- `1` — open

Properties:

`open_flags` —
: Flags showing whether the safe is open and/or unlocked. The 1's bit encodes open (1=open, 0=closed) while the 2's bit encodes unlocked (2=unlocked, 0=locked).

`key_hi` —
: The high byte of the key number to lock/unlock the safe

`key_lo` —
: The low byte of the key number to lock/unlock the safe

Command Behavior:

Do:
`adjacentOpenCloseContainer`

Go:
`goTo`

Get:
`goToAndPickFromIfOpen`

Put:
`goToAndDropIntoIfOpen`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

security device

Description:

Another miscellaneous gadget.

Function:

Provides a way to make a region safe from intrusion.

Notes:

The security device provides a way to protect your avatar and property from harm. When the security device is turned on, no one else can enter or leave the region it is in. It only works in selected regions (like hotel rooms). The device indicates its operation by means of both an indicator light and a humming sound. Note: this object is not currently operational.

Class: `63`

Styles:

- `0` — security device
  - `0` — light off
  - `1` — light on
  - `2` — light blinking

Properties:

`on` —

Flag that the device is turned on.

Command Behavior:

Do:

If the player is holding the device, turn it on or off. Otherwise, `depends`.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

sensor

Description:

Another miscellaneous gadget.

Function:

Tells some property of a region, object or avatar.

Notes:

The sensor can scan for information which is not always directly perceptible to an avatar or, for that matter, to the player's home system. What it scans for depends on what type of sensor it is. For example, a weapon sensor will tell if any of the avatars in the region is carrying a weapon, even if the weapon is inside a bag or similar container. The sensing is performed by the host, so it is guaranteed to be "honest".

Class: `64`

Styles:

- `0` — sensor
  - `0` — lights off
  - `1` — lights on
  - `2` — lights blinking

Properties:

`scan_type` —

What this sensor senses.

Command Behavior:

Do:

If the player is holding the device, operate its sensing function. Otherwise, `depends`.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

sex changer

Description:

A bizarre device.

Function:

Enables Avatars to change from male to female and back.

Class: `90`

Styles:

`0` — sex changer

- `0` — quiescent
- `1` — operating

Properties:

None.

Command Behavior:

Do:

If the player is standing next to the device, change the player's sex (with appropriate histrionics); otherwise, `depends`.

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

short sign

Description:

A sign.

Function:

Conveys information.

Notes:

A short sign is like a sign, but its text is limited to 10 characters (this cuts down on bandwidth requirements, since most signs are short).

Class: 56

Styles:

- 0 — small rectangular sign
  - 0 — rectangle, black text
  - 1 — no rectangle, black text
  - 2 — rectangle, blue text
  - 3 — no rectangle, blue text
  - 4 — rectangle, pink text
  - 5 — no rectangle, pink text
- 1 — hanging needle-point sign
  - 0 — entire sign
  - 1 — without hanger
  - 2 — without needlepoint decoration
  - 3 — without hanger or needlepoint decoration
- 2 — diamond traffic sign
  - 0 — with post
  - 1 — without post
- 3 — hanging tavern sign
  - 0 — entire sign
  - 1 — without post
  - 2 — without hangers or post
  - 3 — post only
- 4 — long, narrow sign
  - 0 — medium length
  - 1 — medium length, shadow text
  - 2 — long
  - 3 — long, shadow text
- 5 — street sign
  - 0 — tall post, normal sign
  - 1 — medium post, normal sign
  - 2 — short post, normal sign
  - 3 — very short post, normal sign
  - 4 — tall post, wide sign
  - 5 — medium post, wide sign
  - 6 — short post, wide sign
  - 7 — very short post, wide sign
  - 8 — no post, normal sign
  - 9 — no post, wide sign
- 5 — highway sign
  - 0 — entire sign, black
  - 1 — entire sign, shadow text
  - 2 — no legs, black
  - 3 — no legs, shadow text
  - 4 — no sign, black text

5 — no sign, shadow text

Properties:

`text` —

What the sign says (10 characters or fewer).

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

shovel

Description:

Your basic digging implement.

Function:

Allows players to open and close holes.

Notes:

The shovel object operates in tandem with the hole object.

Class: `89`

Styles:

`0` — shovel

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

If the player is pointing at a hole, open or close the hole.

Asynchronous actions:

None.

Object:

sign

Description:

A sign.

Function:

For public safety and information.

Class: `57`

Styles:

- `0` — small rectangular sign
  - `0` — rectangle, black text
  - `1` — no rectangle, black text
  - `2` — rectangle, blue text
  - `3` — no rectangle, blue text
  - `4` — rectangle, pink text
  - `5` — no rectangle, pink text
- `1` — hanging needle-point sign
  - `0` — entire sign
  - `1` — without hanger
  - `2` — without needlepoint decoration
  - `3` — without hanger or needlepoint decoration
- `2` — diamond traffic sign
  - `0` — with post
  - `1` — without post
- `3` — hanging tavern sign
  - `0` — entire sign
  - `1` — without post
  - `2` — without hangers or post
  - `3` — post only
- `4` — long, narrow sign
  - `0` — medium length
  - `1` — medium length, shadow text
  - `2` — long
  - `3` — long, shadow text
- `5` — street sign
  - `0` — tall post, normal sign
  - `1` — medium post, normal sign
  - `2` — short post, normal sign
  - `3` — very short post, normal sign
  - `4` — tall post, wide sign
  - `5` — medium post, wide sign
  - `6` — short post, wide sign
  - `7` — very short post, wide sign
  - `8` — no post, normal sign
  - `9` — no post, wide sign
- `5` — highway sign
  - `0` — entire sign, black
  - `1` — entire sign, shadow text
  - `2` — no legs, black
  - `3` — no legs, shadow text
  - `4` — no sign, black text
  - `5` — no sign, shadow text

Properties:

`text` —

What the sign says (40 characters or fewer).

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

sky

Description:

The basic background above the horizon.

Function:

Can be pointed at, returning a location.

Class: 69

Styles:

- 0 — tall ground rectangle
- 1 — short ground rectangle
- 2 — “mod” pattern wall
- 3 — short wall rectangle
- 4 — plain wall
- 5 — brick wall
- 6 — “mod” sky
- 7 — medium sky rectangle
- 8 — mountain
  - 0 — with snowcap
  - 1 — without snowcap
  - 2 — snowcap only
- 9 — cave mouth
  - 0 — rounded top
  - 1 — square top
- 10 — ball
- 11 — tree/cloud
  - 0 — entire cloud
  - 1 — upper right corner
  - 2 — lower left corner
  - 3 — right half
  - 4 — bottom half
- 12 — pipes
  - 0 — full regalia
  - 1 — horizontal pipe section
  - 2 — horizontal section with fitting
  - 3 — horizontal section with upward elbow on left
  - 4 — L-shaped secton
  - 5 — vertical section with elbow at bottom
  - 6 — vertical section with fitting
  - 7 — horizontal fitting only
  - 8 — vertical fitting with small section
  - 9 — vertical elbow only
  - 10 — vertical section with elbow
  - 11 — vertical fitting only
  - 12 — horizontal section with corner on right
  - 13 — valve
  - 14 — vertical section with fitting
  - 15 — right corner
  - 16 — short vertical section
  - 17 — vertical elbow

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goToSky`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

spray bottle

Description:

A body paint sprayer.

Function:

Allows players to change the color/texture of different parts of their Avatars' bodies.

Class: `95`

Styles:

`0` — body sprayer

Properties:

None.

Command Behavior:

Do:

Changes the color pattern of the indicated part of the Avatar's body to that of the sprayer itself.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`noEffect`

Asynchronous actions:

None.

Object:

street

Description:

Your basic roadway.

Function:

Can be walked on.

Notes:

Each street object is a portion of pathway pattern. It behaves just like ground.

Class: `153`

Styles:

`0` — street

- `0` — cross
- `1` — down
- `2` — up
- `3` — left
- `4` — right
- `5` — vertical
- `6` — horizontal
- `7` — left-down
- `8` — right-down
- `9` — left-up
- `10` — right-up
- `11` — left T
- `12` — right T
- `13` — down T
- `14` — up T

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goToCursor`

Get:

`noEffect`

Put:

`goToAndDropAt`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

streetlamp

Description:

Conventional streetlamp.

Function:

Scenic element. Provides light at night.

Notes:

The Habitat can follow a cycle of day and night. When it is night-time, the screen is displayed in darkened colors, which makes things hard to see. Having a streetlamp in the region makes the region appear like daytime, even if it is night elsewhere.

Class: `154`

Styles:

`0` — old-fashion streetlamp
`1` — modern streetlamp

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`noEffect`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

stun gun

Description:

Looks like a gun.

Function:

Delays other players and gives you time to do stuff to them or to escape from them.

Notes:

The stun gun operates like a gun. However, instead of killing or wounding the target Avatar, it "stuns" them. When an Avatar is stunned, it cannot walk for three tries and cannot use weapons.

Class: `91`

Styles:

`0` — stun gun

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

Stun the designated Avatar.

Asynchronous actions:

None.

Object:

super trapezoid

Description:

A fancy background object.

Function:

Allows the definition of very sophisticated backgrounds.

Notes:

A super trapezoid is like a regular trapezoid, except that instead of using one of the standard color patterns, it carries its own color/pattern information with it. Additionally, this pattern may be larger and fancier than any of the standard ones, so we can create elaborate textile-like patterns of various sorts.

Class: `92`

Styles:

`0` — super trapezoid

Properties:

`trapezoid_type` —
What sort of trapezoid this is: 0=sky, 1=wall, 2=ground, 3=impassable.

`upper_left_x` —
The upper left X coordinate of the trapezoid.

`upper_right_x` —
The upper right X coordinate of the trapezoid.

`lower_left_x` —
The lower left X coordinate of the trapezoid.

`lower_right_x` —
The lower right X coordinate of the trapezoid.

`height` —
The height of the trapezoid.

`pattern_x_size` —
The width of the repeating texture map.

`pattern_y_size` —
The height of the repeating texture map.

`pattern(32)` —
The texture map (32 bytes of 2-bit pixels).

Command Behavior:

Do:
`depends`

Go:
`flatGo`

Get:
`noEffect`

Put:
`goToAndDropAt`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

switch

Description:

A switch that you can throw.

Function:

Generic magic talisman.

Notes:

Switches operate rather like magic wands, except that they are immobile. They can be used both for magic and for machinery, though the underlying mechanism is the same in either case.

Class: `97`

Styles:

- `0` — ball
- `1` — switch
  - `0` — switch panel, off
  - `1` — switch panel, on
  - `2` — single switch, off
  - `3` — single switch, on
  - `4` — knife switch, off
  - `5` — knife switch, on
- `2` — dials
  - `0` — round dial, indicator left
  - `1` — round dial, indicator center
  - `2` — round dial, indicator right
  - `3` — meter dial, indicator left
  - `4` — meter dial, indicator center
  - `5` — meter dial, indicator right
- `3` — “flaming” bush
- `4` — spindly bush
  - `0` — without leaves
  - `1` — with leaves
  - `2` — with blinking leaves
- `5` — mushrooms
  - `0` — patch of 4 mushrooms
  - `1` — group of 3 mushrooms on left
  - `2` — single mushroom on right
- `6` — old-fashion streetlamp
- `7` — medium rock
- `8` — large rock
- `9` — pond
  - `0` — entire pond
  - `1` — water only
  - `2` — water and stone
  - `3` — water and grass
  - `4` — stone only
  - `5` — grass only
- `10` — flag
  - `0` — phase A
  - `1` — phase B
  - `2` — phase C
- `11` — coke machine
- `12` — plant type #3

```
13 —
      deciduous tree
  0 — entire tree
  1 — trunk only
  2 — foliage lower right corner
  3 — foliage right half
  4 — foliage upper half
14 —
      evergreen tree
  0 — entire tree
  1 — trunk only
  2 — foliage only
  3 — foliage lower left portion
  4 — foliage lower right portion
15 —
      palm tree
  0 — entire tree
  1 — lower 2/3 of trunk
  2 — upper 1/3 of trunk
  3 — fronds
  4 — whole trunk only
16 —
      dead tree
  0 — entire tree
  1 — trunk
  2 — upper branches
  3 — center branches
  4 — right branches
17 —
      large mushroom
  0 — entire mushroom
  1 — without shadow
  2 — top only
  3 — stalk only
18 —
      cactus
  0 — entire cactus
  1 — barrel only
  2 — left branch
  3 — branches only
  4 — barrel without shadow
19 —
      round tree
  0 — entire tree
  1 — trunk only
  2 — foliage only
20 —
      pipes
  0 — full regalia
  1 — horizontal pipe section
  2 — horizontal section with fitting
  3 — horizontal section with upward elbow on left
  4 — L-shaped secton
```

`5` — vertical section with elbow at bottom
`6` — vertical section with fitting
`7` — horizontal fitting only
`8` — vertical fitting with small section
`9` — vertical elbow only
`10` — vertical section with elbow
`11` — vertical fitting only
`12` — horizontal section with corner on right
`13` — valve
`14` — vertical section with fitting
`15` — right corner
`16` — short vertical section
`17` — vertical elbow

Properties:

`magic_type` —
Sort of magic this switch contains.

`magic_data` —
A 32-bit number interpreted by the magic routines. It's meaning varies depending on `magic_type`.

Command Behavior:

Do:
`If adjacent to the switch, activate its (host defined) magical function, whatever that may be. If not adjacent, depends.`

Go:
`goTo`

Get:
`noEffect`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

table

Description:

A common table.

Function:

Can support things (hold them off the floor).

Notes:

The table is a form of container whose contents are always visible and accessible (i.e., it is permanently "open"). Things can be placed on it and picked up off of it. The table can hold up to 5 items.

Class: `155`

Styles:

- `0` — table
  - `0` — entire table
  - `1` — legs only
- `1` — countertop
- `2` — display case
  - `0` — entire case
  - `1` — without legs
  - `2` — without shelf
  - `3` — without center shelf

Properties:

`open_flags` —

Flags showing whether the table is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndPickFrom`

Put:

`goToAndDropInto`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

teleport booth

Description:

Like a phone booth, but it carries all of you instead of just your voice.

Function:

Zaps avatars and their possessions elsewhere instantaneously.

Notes:

The teleport booth is fairly complex. However, it operates very much like a pay phone. The only difference is that you get transported to the destination.

Class: `74`

Styles:

`0` — teleport booth
- `0` — quiescent
- `1` — active

Properties:

`state` —
The state of the booth: 0=inactive, 1=active.

`take` —
The cumulative income generated by this booth. This value should not be altered manually.

`address` —
The teleport address of this booth.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`noEffect`

Put:
If the item in hand is a token, and the denomination is greater than the cost of a teleport trip, activate the teleport and deduct the price from the token. Otherwise, `noEffect`.

Talk:
If the Avatar is standing in the booth and the booth is active, interpret the string typed as a teleport address and teleport the player there.

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

ticket

Description:

A ticket to the show.

Function:

Your avatar's admission to special events.

Notes:

Tickets are used to control admission to certain regions. You have to have the ticket that corresponds to an event in such a regions in order to be let in.

Class: `75`

Styles:

`0` — ticket

Properties:

`event` —

A number indicating the event this ticket is admission for.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

tokens

Description:

Filthy lucre.

Function:

Money in the Habitat.

Notes:

The standard unit of currency in the Habitat is the Token. On one side it says *Good For One Fare*. On the other side it says *Fiat Lucre*. A single token object can denote any amount of money. The `denomination` property tells how much money a token is currently representing. Certain machines in the Habitat world are coin operated. Dropping a token into them will cause the appropriate amount of money to be consumed. If there is any left over it is left in the avatar's hand. Other sorts of transactions require that the quantity be specified by the player.

Class: `76`

Styles:

`0` — token

Properties:

`denom_lo` —

The low-byte of the token's denomination: how much money this token represents.

`denom_hi` —

The high-byte of the token's denomination.

Command Behavior:

Do:

Display the denomination of the token.

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

If the indicated object is an avatar to whom the player is adjacent, prompt the player for a number of tokens and then pay the designated avatar that many tokens from the amount in hand. This will fail, of course, if the player specifies more tokens then he is carrying at the time.

Asynchronous actions:

None.

Object:

trapezoid

Description:

A generic background object.

Function:

Allows the definition of plain backgrounds.

Notes:

A trapezoid is a general-purpose background object. We use trapezoids because they are more versatile than mere rectangles, but to not have the overhead associated with arbitrary polygons. The trapezoid's properties define its size and shape, and it may be colored/textured with any of the standard patterns.

Class: `87`

Styles:

`0` — plain trapezoid

`1` — rock-pattern trapzoid rock

Properties:

`trapezoid_type` —
What sort of trapezoid this is: 0=sky, 1=wall, 2=ground, 3=impassable.

`upper_left_x` —
The upper left X coordinate of the trapezoid.

`upper_right_x` —
The upper right X coordinate of the trapezoid.

`lower_left_x` —
The lower left X coordinate of the trapezoid.

`lower_right_x` —
The lower right X coordinate of the trapezoid.

`height` —
The height of the trapezoid.

Command Behavior:

Do:
`depends`

Go:
`flatGo`

Get:
`noEffect`

Put:
`goToAndDropAt`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

tree

Description:

Your basic tree.

Function:

Scenic element. Obstruction.

Notes:

This is a fairly inert scenic element, provided almost entirely for visual appeal.

Class: `156`

Styles:

- `0` — deciduous tree
  - `0` — entire tree
  - `1` — trunk only
  - `2` — foliage lower right corner
  - `3` — foliage right half
  - `4` — foliage upper half
- `1` — evergreen tree
  - `0` — entire tree
  - `1` — trunk only
  - `2` — foliage only
  - `3` — foliage lower left portion
  - `4` — foliage lower right portion
- `2` — palm tree
  - `0` — entire tree
  - `1` — lower 2/3 of trunk
  - `2` — upper 1/3 of trunk
  - `3` — fronds
  - `4` — whole trunk only
- `3` — dead tree
  - `0` — entire tree
  - `1` — trunk
  - `2` — upper branches
  - `3` — center branches
  - `4` — right branches
- `4` — large mushroom
  - `0` — entire mushroom
  - `1` — without shadow
  - `2` — top only
  - `3` — stalk only
- `5` — cactus
  - `0` — entire cactus
  - `1` — barrel only
  - `2` — left branch
  - `3` — branches only
  - `4` — barrel without shadow
- `6` — round tree
  - `0` — entire tree
  - `1` — trunk only
  - `2` — foliage only
- `7` — log
  - `0` — entire log
  - `1` — left half
  - `2` — right half

8 — fire hydrant
9 — tree/cloud
  0 — entire cloud
  1 — upper right corner
  2 — lower left corner
  3 — right half
  4 — bottom half
10 —
deciduous tree type #2
  0 — entire tree
  1 — trunk only
  2 — foliage only
  3 — foliage upper left
  4 — foliage upper right
11 —
grass
  0 — tall and short
  1 — tall only
  2 — short only

Properties:

None.

Command Behavior:

Do:
`depends`

Go:
`goTo`

Get:
`noEffect`

Put:
`noEffect`

Talk:
`broadcast`

Reversed Do:
`illegal`

Asynchronous actions:

None.

Object:

vendroid front

Description:

The visible portion of a vending machine.

Function:

Sells stuff in the absence of a human shopkeeper.

Notes:

The vending machine is our general mechanism for distributing goods to players. The vendroid front object operates in tandem with the vendroid interior object. These two classes are always found in pairs. The vendroid front contains a list of items that can be displayed, along with their prices. It also has a property which say which of those items is currently on display. The vendroid interior contains two objects: the vendroid front and the actual item on display. When we change the item on display, we move items between the two parts of the vendroid.

Class: `85`

Styles:

- `0` — store vendroid front
  - `0` — machine with sign panel
  - `1` — machine without sign panel
  - `2` — wall mount without sign panel
  - `3` — wall mount with sign panel
- `1` — news vendroid front

Properties:

`open_flags` —

Flags showing whether the vendroid is open and/or unlocked. Should be permanently set to 3 (open and unlocked). _price — The price of the item currently on display. — A number from 0 to 9 indicating which item is currently on display.

`prices(0:9)` —

The prices of the items in the contents list.

Command Behavior:

Do:

"Rotate" the vending machine, moving the item currently on display from the vending machine interior to its place in the `contents` list, moving the next item in the `contents` list into the display slot (incrementing `currentItem` as we do this), and displaying the price of the new item. The Avatar must be adjacent to the machine to do this. If he is not, he will walk to it.

Go:

`goTo`

Get:

`noEffect`

Put:

Purchases the item currently on display. The Avatar must be holding a token object of sufficient denomination to pay for the item. The price of the item is deducted from the tokens in hand and a clone of the item on display appears on the ground in front of the machine.

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

vendroid interior

Description:

The other part of a vending machine.

Function:

Helps us sell stuff.

Notes:

See the description of the vendroid front object — it does all the work.

Class: `86`

Styles:

- `0` — store vendroid interior
  - `0` — entire machine
  - `1` — glass panel only
  - `2` — glass panel and drop slot
- `1` — news vendroid interior
  - `0` — with "NEWS" sign
  - `1` — without sign
  - `2` — with blank sign
  - `3` — without sign but with sign mounting

Properties:

`open_flags` —

Flags showing whether the vendroid is open and/or unlocked. Should be permanently set to 3 (open and unlocked).

Command Behavior:

Do:

`illegal`

Go:

`illegal`

Get:

`illegal`

Put:

`illegal`

Talk:

`illegal`

Reversed Do:

`illegal`

Asynchronous actions:

None

Object:

wall

Description:

An interior or exterior wall section.

Function:

Graphic element in buildings. Obstruction.

Notes:

A wall is just an opaque rectangle that sits edgewise on the ground. The edge that sits on the ground defines an impenetrable linear barrier. The only way past a wall is to go through a door (or go around it if it is small, of course).

Class: `80`

Styles:

- `0` — tall ground rectangle
- `1` — short ground rectangle
- `2` — "mod" pattern wall
- `3` — short wall rectangle
- `4` — plain wall
- `5` — brick wall
- `6` — "mod" sky
- `7` — medium sky rectangle
- `8` — mountain
  - `0` — with snowcap
  - `1` — without snowcap
  - `2` — snowcap only
- `9` — cave mouth
  - `0` — rounded top
  - `1` — square top
- `10` — tree/cloud
  - `0` — entire cloud
  - `1` — upper right corner
  - `2` — lower left corner
  - `3` — right half
  - `4` — bottom half
- `11` — pipes
  - `0` — full regalia
  - `1` — horizontal pipe section
  - `2` — horizontal section with fitting
  - `3` — horizontal section with upward elbow on left
  - `4` — L-shaped secton
  - `5` — vertical section with elbow at bottom
  - `6` — vertical section with fitting
  - `7` — horizontal fitting only
  - `8` — vertical fitting with small section
  - `9` — vertical elbow only
  - `10` — vertical section with elbow
  - `11` — vertical fitting only
  - `12` — horizontal section with corner on right
  - `13` — valve
  - `14` — vertical section with fitting
  - `15` — right corner
  - `16` — short vertical section

`17` — vertical elbow

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goToCursor`, but the Y-coordinate is clamped at the wall's base.

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.

Object:

windup toy

Description:

Your basic windup doll or robot.

Function:

You wind it up and it walks across the floor.

Notes:

Some day we'd like the windup toy to be able to walk on its own. Right now it is more like a knick-knack.

Class: `82`

Styles:

- `0` — windup doll
  - `0` — still
  - `1` — walking
- `1` — windup penguin
  - `0` — still
  - `1` — walking

Properties:

`wind_level` —
how wound up this doll is.

Command Behavior:

Do:

If the player is holding the doll, increment its wind count (to a maximum of 4).

Go:

`goTo`

Get:

`goToAndGet`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`throw`

Asynchronous actions:

None.

Object:

window

Description:

A conventional house window.

Function:

Graphic element in buildings.

Notes:

Window objects are used to decorate the outside of buildings. They are rather inert.

Class: `157`

Styles:

- `0` — window type #1
  - `0` — whole window
  - `1` — without sash
  - `2` — without left side
  - `3` — without left side, sash
  - `4` — without left side, bottom
  - `5` — without left side, sash, bottom
  - `6` — without right side, bottom
  - `7` — without right side, sash, bottom
- `1` — window type #2
  - `0` — entire window
  - `1` — lower part
  - `2` — upper part
- `2` — crack
  - `0` — medium
  - `1` — very large
  - `2` — large
  - `3` — small
- `3` — barred window
  - `0` — with backing
  - `1` — without backing
  - `2` — backing only

Properties:

None.

Command Behavior:

Do:

`depends`

Go:

`goTo`

Get:

`noEffect`

Put:

`noEffect`

Talk:

`broadcast`

Reversed Do:

`illegal`

Asynchronous actions:

None.