Prolix
A Text-based Participant System for VR

by

Mark Takacs

A thesis submitted in partial fulfillment of
the requirements for the degree of

Master of Science in Engineering

University of Washington

1993

Approved by____
(Chairperson of Supervisory Committee)

College of Engineering (Inter-Engineering)

August 17, 1993

University of Washington

Abstract

Prolix
A Text-based Participant System for VR

by Mark Takacs

Chairperson of Supervisory Committee: Professor Thomas A. Furness III
Engineering (Inter-Engineering)

Virtual Reality has captured the imagination of many people, yet few have experienced it directly due to the currently expensive display and sensor technologies. Those who have experienced virtual worlds often comment on the world's inability to maintain interest with prolonged use. The only thing that remains consistently interesting are other human users. Yet the number of users in a multi-participant world is realistically limited by the great cost of equipping each user with motion tracking sensors, head mounted audio and visual displays, input devices or other interface gear.

This thesis presents a program called Prolix which allows access to the virtual worlds running at the Human Interface Technology Lab (HITLab) without the expensive interface equipment. Prolix assumes only a keyboard and a screen with vt100 capability. Input to the virtual world is accomplished via typed commands and cursor keys, while output appears as narrative text and a simple ASCII-character plan-view of the participant's immediate surroundings. The text interaction style uses the conventions and interface feel of internet Multi User Domains. (MUDs)

An informal pilot study shows that Prolix provides a usable, effective and enjoyable way to communicate with others using the same access interface. Prolix successfully placed a text based person in two VR worlds, one with another immersive user, the other with another Prolix user. However, in the immorsive setting, the Prolix user's graphical display was soon overwhelmed by the volume of information generated by the moving entities and needed to be frequently refreshed.

TABLE OF CONTENTS

LIST OF FIGURES

BIBLIOGRAPHY

[1] Amy Bruckman (asb@media lab.media.mit.edu). Identity workshop: Emergent social and psychological phenomena in text-based virtual reality. Technical report, MIT Media Laboratory, April 1992. Electronic Text available via anonymous ftp from beta.xerox.com:/pub/MOO/papers/identity-workshop.ps.Z.

[2] Elizabeth M. Reid (emr@munan.ee.mu.oz.au). Electropolis: Communication and community on internet relay chat. Technical report, University of Melbourne, 1991. Honor's thesis for Dept of History. Electronic document available via anonymous ftp from beta.xerox.com:/pub/MOO/papers/electropolis.ps.Z.

[3] Jill Serpentelli. Conversational structure and personality correlates of electronic communication. Technical report, Haverford College, 1992. Electronic document available via anonymous ftp from beta.xerox.com:/pub/MOO/papers/conv-structure.txt.Z.

[4] Steve Steinberg, editor. *InterTek*, volume 3(3). Steve Steinberg Press, Winter 1992.

Chapter 1

PROLIX USER GUIDE

This appendix serves as program documentationa and a user's guide. It is divided into several sections, each one addressing a specific aspect of Prolix. The first explains the general concept behind Prolix, while the second explains the choice of interface style. The screen layout is explained, followed by a section on the basic use of Prolix. Section 1.5.1 contains an alphabetical list of the commands, as well as a topical sorting of the commands. Section 1.6 lists and explains all the commands.

1.1 General Concept

Prolix is intended to be a participant system that allows entry into virtual worlds traditionally only accessible with fully inclusive participant systems. The users of Prolix interact with the virtual world through the keyboard and cursor keys, and receive display information through the screen, using only ASCII/vt100 capabilities. It is hoped that a wider range of people will be able to experience some semblance of VR if the interface avoids the higher end graphics systems such as X-Windows, or the complications of various specialized home computer platforms.

1.2 Interface Style

A goal of the HITLab is to create a virtual community with its own personality and flavor. Yet most of the worlds created, while interesting, are sterile and museum-like. None of these worlds yet created have the style and feel of a community such as a comfortable local cafe, bar, or even the sense of electronic community found on many MUDs or IRC [1], [2], [3], [4]. What's missing are the *people*. Because Prolix's hardware requirements are so commonly available, I hope that its widespread use will encourage many people to enter the current VR worlds. Having many people present and communicating within the same virtual space should allow the space to acquire an identity associated with the people who frequent that space.

1.3 Screen Layout

The typical Prolix screen is split into two major sections, the graphical display and the text display. Each of which is further divided in two: the graphics display has a plan and side view, while the text display has a separate input and output window. (See figure 1.1)

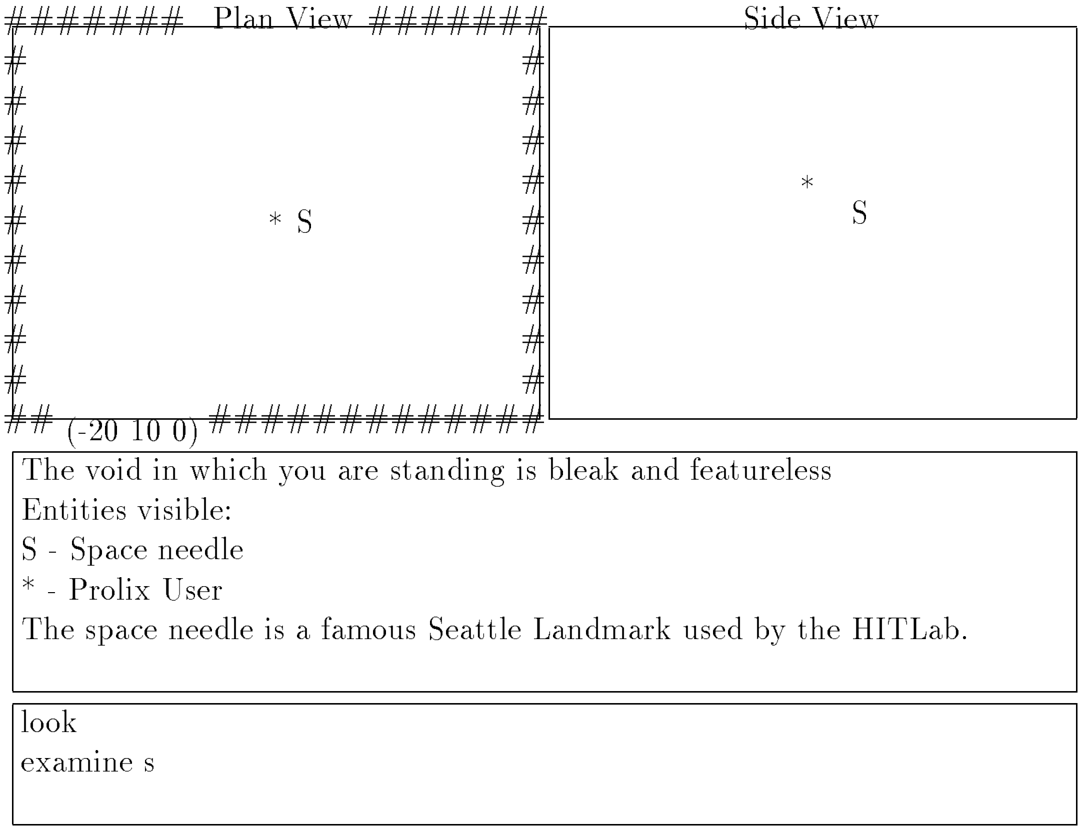


Figure 1.1: The text display

1.3.1 Text Display

In its initial configuration, the text display takes up the bottom half the screen, with three to five lines (adjustable at compile time) at the bottom of the screen allocated to echo the user's typing input. (See figure 1.1) The text output appears in the output windows and scrolls upward until it reaches the top of the output window.

The default output window size ends at the beginning of the graphics window. It is possible to eliminate the graphics window entirely, in which case the output window expands to encompass the entire window. In this mode, the up cursor key recalls the last typed line of input.

The results of user issued commands, text output from events in the VR world, and behaviors and communications from other participants and entities appear in the output window. To avoid cluttering the display, Prolix doesn't echo the user's commands to the output window, just their results or a feedback message.

1.3.2 Graphic Display

In its initial configuration the graphics window takes up the top half of the screen. (See figure 1.2) The left half of the display shows the plan view, while the right side displays an orthographic side view. In the default configuration, each discrete character-wide space on the display grid represents a 1x1x1 meter volume. A single character icon is placed on the screen according to the coordinates of the object in the VR world. The plan view window shows the entities' `x` and `z` coordinates, while the side view window shows the missing `y` coordinate as well as the `x` coordinate. The HITLab's VR worlds follow a protocol where the center of the world is defined te be (0,0,0), with `x` as left and right, `y` as up and down, and `z` as depth into the scene. Mentally combining the information provided by the two views allows you to place the object in the world, as well as its elevation or altitude. If an object moves, its icon moves in both windows simultaneously.

In figure 1.2 the "*" represents the user, who is located slightly to the left of and above the "S", which is the base of the space needle at the center of the VR world coordinates(0,0,0). The user and the space needle are separated by at least 20 meters horizontally in this example.

In the default configuration, your icon is free to move about the display until it reaches a configurable radius from the center of the display, at which point the display is re-centered about the user. The other mode fixes your icon in the center of the display and any movement commands cause the world to go scrolling by.

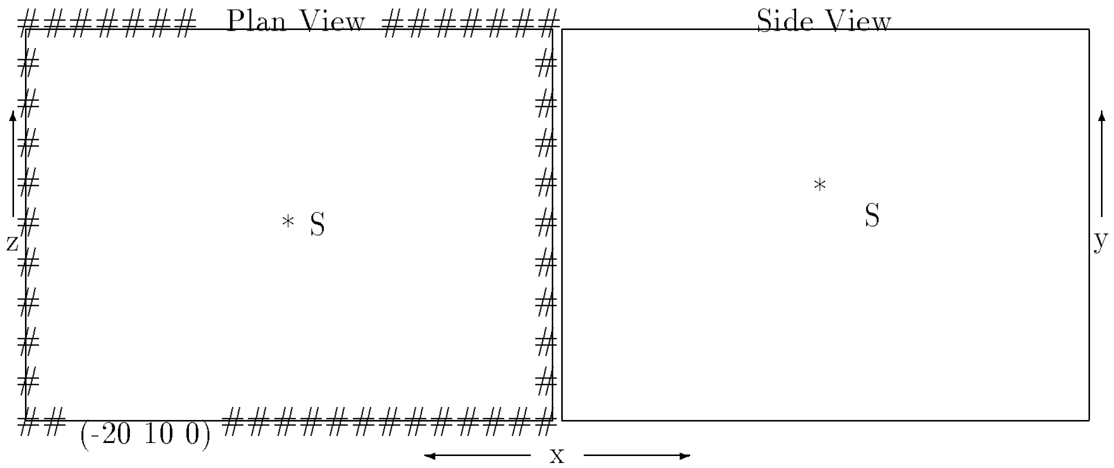


Figure 1.2: The graphics display

1.4 Operation Guidelines

This section gives a brief description of some of the more common commands and their intended uses.

1.4.1 Navigation

Navigation through the VR world with Prolix is accomplished by the use of the cursor keys. Prolix starts with the plan view graphical display as the active window. The active graphical display window is outlined with "#" marks, rather than dashes.

For the sake of clarity, I will assume you are in a world where the plan view represents looking down on a space where the top of the screen is north, bottom is south, and left and right are west and east. The side view has the top of the screen as up, the bottom of the screen as down, and the left and right sides as west and east. See figure 1.3.

When you hit the cursor-right key, your icon moves one space east in each of the windows. If you use the up-cursor or down-cursor while the plan view is active you will move north or south, respectively. However, if the side view is active, pressing the up-cursor or down-cursor moves your icon up or down in elevation. In figure 1.3 pressing the up-cursor will move you north. When your icon comes near the edge of the screen, you are repositioned in the center of the display and the screen is redrawn. The percentage of the screen on which you can move around is a user preference and

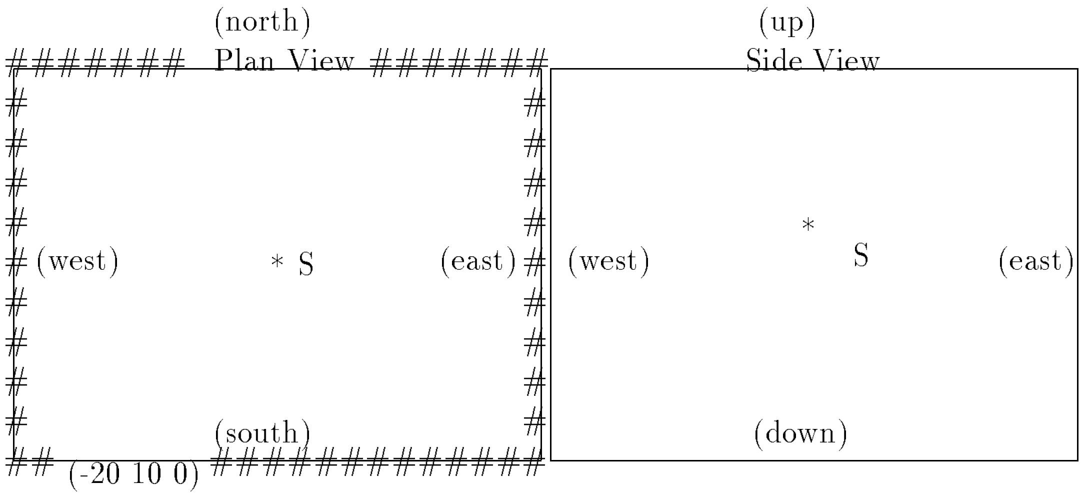


Figure 1.3: Movement orientation

is set with a command called `sweet` and defaults to 100%, which determines how close your icon can get to the edge of the screen before being repositioned. Setting this value at 0% causes the user's position to be fixed at the center of the screen, where subsequent use of the cursor keys causes the screen to scroll past in that direction.

A teleportation command called `goto` is provided to allow you to instantly jump to some known location. Providing the name or icon of another entity as arguments to this command moves you directly to that entity. If no argument is provided, you are moved to the absolute world coordinates (0,0,0).

1.4.2 Communication

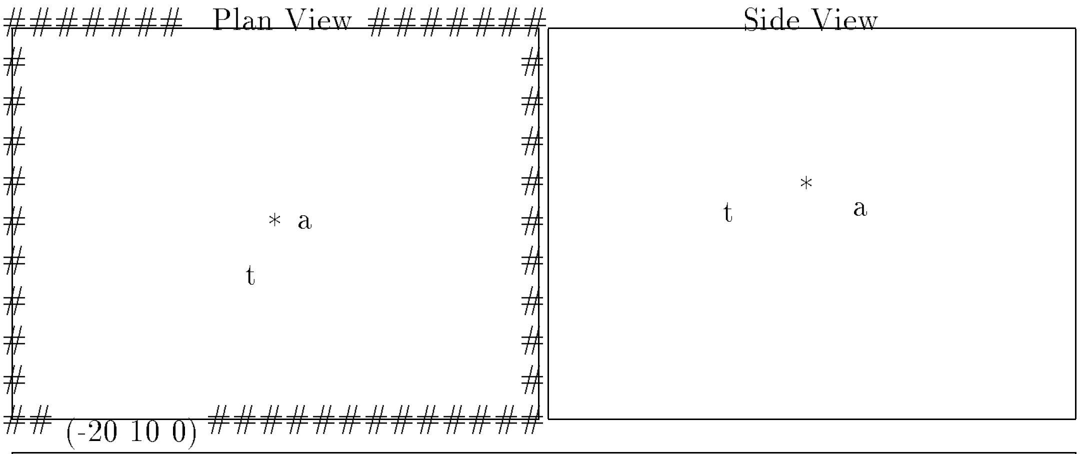
Communication in the VR world with Prolix is accomplished mainly through the `say` and `tell` commands. The `say` command causes anything you type to be passed out into the VR world for other entities to see and/or process. Other entities receive information on who sent the message and the message's content. It is up to the particular entity to decide how to process this information. Any Prolix interface will automatically display incoming text messages.

An example session will serve to illustrate the essential points. In figure 1.4, the user has issued the `who` command, (line (a) in figure 1.4) which gives a listing of all the participant systems in the VR world (the line marked with (1)).

Then the user decides to ask a question using the `say` command (b) so that it is distributed to everyone. Appropriate feedback (2) is given in the output window.

The user known as Teklon replies with a behavior (3) known as an `emote`. The `emote` command prepends whatever text you type with your name. In this example, Teklon has typed `emote shrugs` to produce the output marked with a (3). In this manner, your character can appear to do many things that lend atmosphere and style to the conversation.

Reading Ari's reply (4), the user decides to reply directly to Ari with the `tell` command (b)(5), which is only received by the person to whom it is addressed (Ari). Other users (like Teklon) remain unaware that anything is even happening. Technically, everyone receives all the messages, but each individual Prolix entity performs filtering functions (see chapter ?? for more details). Ari replies in kind (6).



(1) 3 Users online: Ari (a), Teklon (t), Tak (*)
(2) You say: Anyone know when the meteor shower starts?
(3) Teklon shrugs.
(4) Ari says: I saw a posting about it.
(5) You tell Ari: I think it starts at midnight..
(6) Ari tells you: I think you're right.

(a) who
(b) say Anyone know when the meteor shower starts?
(c) tell ari I think it starts at midnight.

Figure 1.4: Communication in Prolix

There are several other ways that Prolix allows you to shape your communication, including `shouts`, `echos`, and `aliases`. These are discussed in section 1.6.

1.4.3 Database Manipulation

Database manipulation is the term for altering the database that you keep about other entities and the information you disseminate to them.

Each VR entity has a collection of attribute-value pairs that can be modified by that entity. Figure 1.5 shows a typical entity. Prolix provides a way to change the traditional values (visibility, name, etc), as well as specific fields recommended by the Prolix protocol.

```
('name' 'space needle')
('visible' t)
('wireframe' nil)
('teleportable' t)
('picture-desc' 'needle')
('alias' 'needle')
('icon' 'S')
('short' 'The Space Needle')
('long'
 'The Space needle is a Seattle icon and a defacto VR
  object as well.  It rises many meters overhead and
  is built from hopefully sturdy supports. Two elevators
  provide access to the observation deck at the top.')
```

Figure 1.5: Example entity

The `icon` command allows you to change the icon that is displayed on the graphics display of other Prolix users. For the benefit of any inclusive users that may be in the world, Prolix provides the `pict` command which takes as an argument the filename of your new graphical description file.

The `name`, `title` and `short` commands allow you to change the information shown to other Prolix users when they ask for information about you. The `name` command allows you to change your name.

Normally when other users `look` or issue a `who`, they are provided with the short description of each entity. The short description is composed from two subfields, the name and title field. For example, the user "tak" with the title "- the lowly peasant" have a short description of "tak - the lowly peasant" which would be displayed as a result of a `who`, `look`, or `examine`. The `short` command enables complete control of your short description. Using the short description, you could appear as "a rock." when people `look` in the room or view the `who` list.

The current version of Prolix allows you to alter anything in your database, including your name, icon and other identifying data. As with any great flexibility, there is usually a compromise somewhere. This great flexibility assumes the users trust other users not to maliciously masquerade as someone else. If prolonged usage of Prolix shows this to be a problem area, the command set can be restricted so that key identification data is unable to be modified.

1.4.4 Interface Configuration

Prolix offers its users the option to customize the interface and get status information about the current state of the program.

As mentioned in section 1.4.1, Prolix has several variables governing how movement in the graphical display works, including a command to control the amount of VR world space each screen grid space represents. This command is called `grid`, and takes a numerical argument that changes the space represented by each grid square. The grid size defaults to one.

Another variable that the user can set is the `sweet` percentage. This is the percentage of the screen (measured from the center) that the user can move about on without causing a screen refresh. The default sweet percentage is 100%, enabling the user to advance to the edge of the screen.

Other commands of interest are the `help` command, which prints a brief (but complete) listing of all commands. The `prolix` command shows a list of various interface command variables such as the `sweet` percentage, the grid size, the active window, your screen and world coordinates, and other miscellaneous information about the displays.

There are several control sequences that are useful during operation. The most useful one is perhaps the refresh (`<control-l>`), which discards all optimization information and draws the screen over again from scratch. There are a number of other control sequences, but the only other one that needs to be mentioned here is the `quit` command, which is triggered by typing `quit` or by the `<control-c>` sequence.

1.5 Command Reference

This section presents several brief lists of the commands grouped in different ways. Section 1.5.1 is an alphabetical list of all possible typed and control sequence commands. Section 1.5.2 is a functional grouping of the commands into the four categories of communication, movement, database manipulation and interface control.

1.5.1 Alphabetical List of Commands

Figure 1.5.1 alphabetically lists thhe complete set of commands that Prolix accepts, including control-key sequences.

Alphabetical List of Commands		
Control-b	emote	quit
Control-c	examine	say
Control-d	goto	shout
Control-l	grid	short
Control-o	help	sweet
Control-u	home	tell
Control-z	icon	title
Cursor Keys	long	visibility
clear	look	who
describe	prolix	wireframe
echo	pict	

Figure 1.6: Alphabetical list of commands

1.5.2 Commands by Function

Figure 1.5.2 contains a functionally grouped list of all the command and control-sequences that Prolix accepts.

Commands by Function			
Communication:	say	tell	shout
	emote	echo	
Movement:	<cursor keys>	goto	home
Database:	who	look	examine
	icon	name|short	pict
	long|describe	title	
	visibility	wireframe	
Interface:	help	brief	sweet
	grid	quit	prolix
	ctrl-l	ctrl-c	ctrl-d
	ctl-o	ctrl-u	ctrl-z
	ctrl-b		

Figure 1.7: Commands by Function

1.6 Description of Commands

This section gives complete specifications of all the Prolix commands. The listings include command syntax, description, an example, and a list of related commands. The commands are logically grouped into subsections for communication, movement, database manipulation and interface control.

1.6.1 Communication

say

Syntax: `say` <*text*>

Description: The `say` command is used to send a message to everyone in the immediate area. The message is automatically prepended with your name. This is the most common form of communication.

Example: you type: `say The metro demo is fun. Let's try it.`
you see: You say: The metro demo is fun. Let's try it.
others see: Tak says: The metro demo is fun. Let's try it.

Abbreviation: '

See Also: tell, shout

tell

Syntax: `tell` <*name*> <*text*>

Description: The `tell` command is used to send a person-to-person message which is automatically prefaced with your name. The message is only seen by the person to whom it is addressed. The recipient will recieve the message even if they are in another area.

Example: you type: `tell ari Meet me at the metro demo.`
you see: You tell Ari: Meet me at the metro demo.
ari sees: Tak tells you: Meet me at the metro demo.
others see: (others don't see anything)

See Also: say, shout

shout

Syntax: `shout` <*text*>

Description: The `shout` command is used to send a message to everyone logged in at the time. The text message you provide is automatically prepended with your name.

Example: you type: `shout Tell me if you want a metro tour.`
you see: You shout: Tell me if you want a metro tour.
others see: Tak shouts: Tell me if you want a metro tour.

See Also: say, tell

echo

Syntax: `echo` <*text*>

Description: The `echo` command causes the text provided by the user to be shown to all other users in your area.

Example: you type: `echo The Metro train pulls into the station.`
you see: The Metro train pulls into the station.
others see: The Metro train pulls into the station.

Abbreviations: ;

See Also: emote

emote

Syntax: `emote` <*text*>

Description: The `emote` command is used to provide atmosphere by sending some text that describes what you're doing. Your name is automatically prepended.

Example: you type: `emote smiles.`
you see: You smiles.
others see: Tak smiles.

Abbreviations: :

See Also: echo

1.6.2 Movement

This section describes the commands that allow you to navigate through the VR world.

Cursor Keys

Syntax: <*cursor keys*>

Description: The cursor keys provide the primary way to move yourself through the virtual world. Each key hit moves you a discreet distance (as set by the `grid` command) in that particular direction.

See Also: goto, home

goto

Syntax: `goto` <*x,y,z coordinates*>

Description: The `goto` command instantly changes your location to the specified coordinates.

Example: you type: `goto 10 20 0`
you see: You move to position (x=10 y=20 z=0)
others see: Tak teleports to a new position.

See Also: home, cursor keys

home

Syntax: `home`

Description: The `home` command instantly changes your position to a predetermined set of coorditates, typically x=0, y=0, z=0.

Example: `home`

See Also: goto, cursor keys

1.6.3 Database Commands

This section contains a description of all the commands that allow you to alter the data stored about you, as well as query for data about the VR world.

look

Syntax: `look` {`at` <*icon*> | <*text*> }

Description: This command has two uses. The first, with no arguement, gives you a list of all the entities in the space with you. This includes other participants as well as inanimate VR world objects. The information provided consists of the entity's icon and short description, which in the case of other participants, will usually consist of their name and title. The second use, given as `look at` is the same as using the `examine` command.

Example: `look at tak`

See Also: examine

examine

Syntax: `examine` <*icon*> | <*text*>

Description: This command displays the "long" description of the entity you request. The argument can be the representative icon, or the entity's name.

Example: `examine tak`

Abbreviations: exa

See Also: look

who

Syntax: `who`

Description: This command shows a listing of all the other participant systems in the current VR world with you, including immersive participants as well as other Prolix users. The information provided is a single line consisting of the character's icon and short description, which usually consists of a name and title.

Example: `who`

See Also: look, examine

icon

Syntax: `icon` <*character*>

Description: This command changes the graphical icon that represents your position in the VR world on other Prolix graphical displays.

Example: `icon T`

See Also: name, title, short, pict

short

Syntax: `short` <*text*>

Description: The `short` command is included primarily for total database control. Text entities have three basic things that identify them: an icon, a short description, and a long description. As a default, the short description is the user's name, followed by their choosen title. If this command is used, the given short description overrides the default combination.

Example: `short This is tak's new short description.`

See Also: icon, name, title, long

title

Syntax: `title` <*text*>

Description: Your title, combined with your name, produces the information shown to people when they `look` around the room or issue a `who` command. Using this command allows you to change your title information.

Example: `title - the great and powerful wizard of Oz`

See Also: name, short

name

Syntax: `name` <*text*>

Description: Use of the `name` command changes the identifier used as your name. Issueing this command automatically changes your icon to the first letter of your name.

Example: `name ted`

See Also: icon, short, title, long.

long

Syntax: `long` <*text*>

Description: This command lets you alter the detailed description that others see when they `look` or `examine` you.

Example: `long Tak is a very tall human with big feet.`

See Also: describe

describe

Syntax: `describe` <*text*>

Description: Use of the `describe` command has the same effect as issuing the command `long`. The two different names exist for the same function to reflect the different ways the command can be pictured. `long` is the database field where the info will be stored, while `describe` is more descriptive of the function.

Example: `describe Tak is a very tall human.`

See Also: long

pict

Syntax: `pict` <*filename*>

Description: This command exists for the benefit of any inclusive users who may be with you. This command changes the polygon picture file associated with your body to the new file that you specify.

Example: `pict cube`

See Also: wireframe, icon

visibility

Syntax: `visibility {t | nil}`

Description: This command toggles your entity's VEOS and Prolix visibility status. Issued as `visibility nil`, this command removes your icon from the graphical display and indicates that your picture should not be shown to other immersive users. `visibility t` restores your icon and picture.

Example: `visibility nil`

See Also: wireframe

wireframe

Syntax: `wireframe { t | nil}`

Description: This command exists for the benefit of any inclusive users who may be with you. Issued as `wireframe t`, it changes your current graphical representation from polygon shaded to a wireframe model. `wireframe nil` has the opposite effect. This has absolutely no effect on other Prolix viewers.

Abbreviations: wire

Example: `wireframe t`

See Also: visibility

1.6.4 Interface Commands

The commands listed in this section help maintain and costomize the Prelix interface. These commands do not effect the VR world at all, with the exception of the quit command, which causes your entity to leave the VR world.

help

Syntax: `help`

Description: This command prints a list of commands ordered by functionality, much like the list in section 1.5.2.

Abbreviations: ?

Example: `help`

quit

Syntax: `quit`

Description: This causes Prolix to close down and exit immediately, as well as exiting any spaces you may be in.

Example: `quit`

See Also: control-c (quit), control-z (suspend)

grid

Syntax: `grid` {<*number*>}

Description: Specifies grid length for each space on the graphical display. If issued without an argument, it reports the current grid value.

Example: `grid 10`newline Sets each space in the graphical display to a 10x10x10 volume.

See Also: sweet, movement - cursor keys

sweet

Syntax: `sweet` {<*percentage*>}

Description: The sweet percentage control how far from the center of the display you can move before a refresh is triggered and you are placed back in the center of the screen. The default value is 100% which triggers a refresh when you reach the edge of the display. A value of 50% would allow you to move halfway to the edge of the screen before recentering and redrawing the display. A value of 1% will mean that you do not move from the center of the screen. As you press the cursor keys, the world scrolls by you. Issuing the command without any value reports the current percentage rate, as well as how many grid spaces this allows you to move with the current screen size.

Example: `sweet 100`

See Also: control-l (refresh), grid

Control-b

Syntax: `<control-b>`

Description: This command toggles the output window from its normal half screen size to a full screen size and back. Note that if the graphics display has been disabled with control-d, you will be unable to regain the half-screen sized window, as that would leave half the screen unused.

Example: `<control-b>`

See Also: control-d (display toggle)

Control-c

Use of the control-c sequence has the same effect as issuing the command `quit`.

Control-d

Syntax: `<control-d>`

Description: This control sequence toggles the graphics display on and off. Note that in the default configuration, the text output window extends only to the edge of the graphics display. If you turn off the graphics display, Prolix expands the text output area to fill the whole screen, just as if the user had issued a control-b. Pressing control-d again will re-enable the graphics display. Note that the text output will remain in "big" mode, with text scrolling under the graphical display.

Example: <`control-d`>

See Also: control-b (BIG toggle)

Control-l

Syntax: <`control-l`>

Description: Throw away the `curses` screen optimization information and redraw the screen from scratch. The graphics display is recentered as well.

Example: <`control-l`>

See Also: clear, sweet

Control-u

Syntax: <`control-u`>

Description: Clears the current text input line.

Example: <`control-u`>

Control-z

Syntax: <`control-z`>

Description: Causes the execution of the program to be suspended. When proceesing is returned to the foreground via the UNIX `fg` command, you will be in FERN mode and able to issue LISP or FERN calls. To resume the Prolix interface, you will need to type `(fern-go)`.

Example: <`control-z`>

clear

Syntax: `clear`

Description: Clears the screen by printing a line of dashes "-" followed by numerous newlines. Use this comand when the refresh command fails to properly remove screen clutter.

Example: `clear`

See Also: control-l (refresh)

history

Syntax: <`cursor-up`> (only in text only mode)

Description: If the graphics display has been turned off with <control-d>, the <up-cursor> operates to recall the last typed command.

Example: <`cursor-up`>

See Also: control-d (display, display toggle)

prolix

Syntax: `prolix`

Description: This command prints a list of information about the interface itself. Including currunt screen size, sweet percentage, grid size, active screen, 6D world coordinates, screen origin in the world and output window size.

Example: `prolix`

See Also: grid, sweet

Chapter 2

SOURCE CODE AVAILABILITY

As the source code for the Prolix library additions to VEOS contains more than 7000 lines of code (courtesy of `wc -l *.c *.h *.l *.y`), it would have been impractical to include it here. Tho source code will be available at the HITLab's forthcoming anonymous ftp site (probably `ftp.hitl.washington.edu`) or directly through me.

I would welcome any comments or information requests. I can be reached at the address below until at least December 1993.

Mark Takacs
University of Washington
c/o Human Interface Technology Lab FJ-15
Seattle, WA 98195
tak@hitl.washington.edu
mtakacs@fry.halcyon.com