Functional specs for Ghu

(God's Handy Utility)
by
Chip Morningstar

Lucasfilm Ltd. Games Division
March 10, 1987

Introduction

This document describes the preliminary specifications for **Ghu**, a utility for Habitat™ operators on the host system. The document is substantially revised from an earlier version.

The specific details of the function set, command syntax and so on are not final at this point. This document is intended to be the starting point for discussion. We anticipate that the design described here will change considerably as it evolves. In particular, Quantum's technical advice and operational policies may lead to considerable changes to the design that follows. Throughout the document, italicized parenthetical comments *(like this one)* indicate possible alternate approaches to things.

What It's For

Ghu is meant to be a full-featured operator's utility for the Habitat system. Many of the functions described here are redundant with **Twiddle**; in fact, the starting point for this definition is the **Twiddle** command set. In many respects, **Ghu** will just be a sort of "**Super Twiddle**". However, a major distinction between **Ghu** and **Twiddle** is that **Ghu** is intended to be capable of dealing with active regions. Like **Twiddle**, **Ghu** is intended to be run from an ASCII terminal connected to the Stratus host by Quantum and Lucasfilm operational personnel (i.e., "Authorized Personnel Only").

Ghu is not intended to be a substitute for the "god" functions that can be made available to privileged users using special magical objects or hidden features in the Commodore 64 software (e.g., Avatar invisibility). These operate from *within* the Habitat environment and are generally tailored to specific tasks. Nor is **Ghu** supposed to offer a major handle on the task of large-scale creation of regions and their contents, even though it will be of significant use in this. The bulk of this task will be dealt with by other utilities. The job of **Ghu** is to provide a *general* mechanism for *managing* the world.

Functional Overview

Ghu is a line-oriented utility. It reads lines from the terminal, parses them into commands, and executes the commands. It is not designed as a visual, CRT-oriented program since it must be usable over something like Telenet. *(It may be that we wish to create an alternate "Visual Ghu" version for use with directly connected terminals. This obviously could be much zippier in its interaction. However, our perspective is from 2500 miles away and we always have to talk to the Stratus through Telenet, which keeps us painfully focused on a line-oriented front-end.)*

Specific design goals of **Ghu** include the following:

- Access to the states of objects in *both* the disk database and the running (i.e., in memory) system.
- Incorporation of knowledge about all the different databases and executing processes that make up the Habitat system.
- Specification of input data in a variety of formats (e.g., hex *and* decimal *and* ASCII strings).

- Definition of symbolic names for frequently used values.
- Allowance for simple expressions wherever values are required.
- "One shot" commands issuable directly on the **Ghu** command line from VOS command level.
- Frequent operations accomplishable with a single command rather than requiring a sequence of commands.
- Incorporation of greater knowledge about specific details of different object properties (for example, it should understand that the high-bit of the Y-coordinate of an object distinguishes a background object from a foreground object, except for objects that are in containers in which case the Y-coordinate is the container slot) to save operator effort and validate data.

One feature in particular deserves special mention: **Ghu** is intended to support virtually all possible manipulations and information retrieval functions associated with the various Habitat databases and the running Habitat environment. Since we do not wish to allow every operator to have unlimited power over everything, **Ghu** is designed with a very finely-grained privilege control mechanism that determines which capabilities are permitted to which accounts.

Privilege Control

This utility is designed to support every conceivable manipulation or query associated with any of the Habitat databases. This makes it enormously powerful, and such power presents not only the security problems associated with deliberate misuse but those associated with accidental mishaps as well. Clearly, this power must be checked somehow.

One possible path to controlling **Ghu**'s power is to break it into a variety of separate utilities, each of which manages a particular aspect of the database. This approach has merit; however, it entails considerable per-operation overhead and does not allow much state information to be carried between successive commands.

The approach we have chosen is to maintain a privileged database, much like a password file, of *permission records* that specify what each **Ghu** user may and may not do. The permissions are very finely grained, so that we can limit particular individuals' control to particular parts of the universe or to particular roles.

For each **Ghu** user, there is a set of global restrictions that limits the regions, Avatars and objects that the user can find out about or change. Specifically, for each of the three major databases (region, object and Avatar) we maintain two *restriction lists*: a *read restriction list* and a *write restriction list*. Each of these lists is an ordered set of pairs of global ID numbers. A user may only access a database record if the global ID of the record is within the range of one of these pairs on the list appropriate to the database and access type sought.

In addition to the global restrictions, there are restrictions associated with each command that limit the use of that command. The nature of these restrictions varies with the command, so they are described below with the commands themselves.

Types, Values and Expressions

In a number of the commands described below, one must provide a global ID number for some entity in one of the databases. Since the key-spaces for the databases overlap (potentially if not in fact), it is necessary to indicate, in addition to the global ID number, what sort of entity it is that one is referring to. Many of the command definitions refer to something called an *obspec*, which is an object specifier. It has the following form:

[*type* **]** *value*

Where:

type is one of the five letters `a` (for Avatar), `o` (for Object), `r` (for Region), `s` (for String) or `u` (for Undefined, i.e., no particular type is intended). If *type* is omitted or is `u`, a default is chosen that is based on the particular type of command being executed. In the definitions below, this default is indicated by a letter after a hyphen ("-") character in the particularly syntactic entity in question. For example, *obspec-r* refers

to an *obspec* whose default type is Region.

value is either a number, a string, a symbolic name (defined using the `name` command, which is described below), or a simple expression.

A number may be specified in any of a number of different radix formats. For the sake of eliminating arguments about which is "better" (and at the expense of slight complication to the spec) we have chosen to adopt both the Unix-derived representations that we use at Lucasfilm and the PL/1 derived representation used by Quantum, since this can be done without significant ambiguity.

- Decimal — a decimal number is represented by one or more decimal digits, the first of which may not be `0` (e.g., `47`, `3`, `123786432333`, etc.)
- Octal — an octal number is represented by zero or more octal digits, preceded by a `0` digit (e.g., `0377`, `022`, `0`, `047`, etc.). Alternatively, a series of octal digits enclosed in apostrophes and followed by `b3` may be used (e.g., `'377'b3`, `'22'b3`, `'0'b3`, `'47'b3`, etc.)
- Hexadecimal — a hex number is represented by one or more hex digits, preceded by `0x` or `0X` (e.g., `0xf000`, `0x80`, `0X10FF`, `0x0`, etc.). Note that case is not significant in the hex digits `a` through `f`. Alternatively, a series of hex digits enclosed in apostrophes and followed by `b4` may be used (e.g., `'f000'b4`, `'80'b4`, `'10ff'b4`, `'0'b4`, etc.)
- Binary — a binary number is represented by one or more binary digits (i.e., `0` or `1`) preceded by `0b` or `0B` (e.g., `0b10101010`, `0B111`, `0b10001`, etc.) Alternatively, a series of binary digits enclosed in apostrophes and followed by `b` may be used (e.g., `'10101010'b`, `'111'b`, `'10001'b`, etc.)
- Quarters — a quarter (base-4 number) is represented by one or more base-4 digits preceded by `0q` or `0Q` (e.g., `0q1230`, `0q2020`, `0Q0013`, etc.) Alternatively, a series of base-4 digits enclosed in apostrophes and followed by `b2` may be used (e.g., `'1230'b2`, `'2020'b2`, `'0013'b2`, etc.)

A string is represented by a series of characters enclosed in apostrophes or quotation marks (e.g., `"hello world."` or `'this is a string'`). Non-printing and special characters may be embedded in the string by using the backslash character ("`\`") as an escape followed by an indicator character (we are borrowing this from Unix). The escape character codes are:

`\n`	newline
`\t`	horizontal tab
`\b`	backspace
`\r`	carriage return
`\f`	form feed
`\e`	escape
`\\`	backslash
`\'`	apostrophe
`\"`	quote
`\^`*c*	CONTROL-*c* (where *c* is any character).
`\`*ddd*	arbitrary byte (where *ddd* is one, two or three octal digits).
`\x`*hh*	arbitrary byte (where *hh* is one or two hexadecimal digits).

A symbolic name is represented as a sequence of alphanumeric characters, the first of which may not be a decimal digit. Names may also contain the characters underscore ("`_`") and dollar sign ("`$`"). The case of alphabetic characters in symbolic names *is* significant.

In general, anyplace where a *value* is called for a simple expression may be given. The expression syntax is:

number
string
name
`-` *expression*
`^` *expression*
`(` *expression* `)`

expression `+` *expression*
expression `-` *expression*
expression `*` *expression*
expression `/` *expression*
expression `&` *expression*
expression `|` *expression*
expression `^` *expression*
obspec-o `.` *field*

The operators all have their usual meanings and precedences: `+`, `-`, `*` and `/` indicate addition, subtraction, multiplication and division respectively; `&`, `|` and `^` represent bitwise AND, OR and XOR (exclusive-OR); unary `-` and `^` indicate arithmetic and bitwise negation. Unary operators take precedence over binary operators and multiplicative operators over additive. Parentheses may be used at any point to control the order of evaluation.

The `.` operator may be used to select the value of a field of an object, Avatar or region. This value will be read from the database, independent of the "working entity". Be careful about using values that have been changed but not updated, since this sort of expression will always yield the old (pre-change) value.

Each term in an expression has a type. The type of a *number* is `u`. The type of a *string* is `s`. The type of a *name* is assigned when the name is defined. The type of a unary operator term or of a parenthesized sub-expression is simply the type of the *expression* being operated upon. The type of a binary operator term is chosen from the types of the two sub-expressions as follows: if the two sub-expressions have the same type, then the result has this type; else if one of the two sub-expressions has type `u`, then the result has the type of the other (non-`u`) sub-expression; otherwise an error results from trying to combine two incompatible types.

Note that since there are no string operators, expressions involving strings per se are rather silly. In arithmetic expressions, we treat a single character string as a number whose value is the value of the 8-bit ASCII code for the character in the string. Similarly, two-character strings represent 16-bit numbers and so on. We do *not* adopt the nearly useless PL/1 convention of trying to convert the string containing the characters `'4'` and `'7'` into the number 47.

The Commands

The specifics of the various commands will be described shortly. First, some general information:

The command names have been chosen so that they are unique in the first few characters. Single letter abbreviations are usually possible. However, any unambiguous initial substring of a command name is an acceptable abbreviation.

In general, commands are terminated by the end of the line, but multiple commands may be given on a line separated by semicolons ("`;`"). A single command can be spread over multiple lines by ending all but the final line with a backslash ("\");

Any command may be followed by

> `>` *filename*

or

> `>>` *filename*

in which case the printed output generated by the command is written to the given file instead of to the terminal. If `>>` was specified, the output is appended to the file; otherwise the output replaces the file's previous contents.

General Ghu Environment Manipulation

`help` [*command*]

If no *command* is given, print out a list of the available commands along with a quick one-liner description of each. If a *command* is given, print out a screen worth of explanation of that command.

Restrictions: none.

`name` *name* [`=`] *obspec-u*

Assigns the symbolic name *name* the type and value specified by *obspec*. The type associated with *name* will be whatever the type of *obspec* happens to be.

Certain names are predefined by **Ghu** automatically. These include all the class numbers, represented by symbols of the form `class_foo`. In addition, any context where an Avatar user ID is expected, user names may be used as if they were defined symbols (though they won't actually be in **Ghu**'s list of predefined symbols since there are far too many of them).

Restrictions: none.

`allnames`

Prints out the names, values, and associated types of all symbolic names currently defined.

Restrictions: none.

`macro` *name* [`=`] *string*

Defines the macro *name* as the string *string*. Whenever the symbol *name* is encountered on a command line, it is textually replaced with *string*. Macros may call other macros (i.e., macro expansion continues until exhausted).

The "pseudo-macro" `$` is automatically defined in `set` commands as the current value of the field being set. In addition, other macros may be defined by a standard definition file that **Ghu** reads at the beginning of execution. These standard macros will be used to encapsulate details about the internal configurations of various object properties.

Restrictions: none.

`allmacros`

Prints out the names and definitions of all macros currently defined.

Restrictions: none.

`evaluate` *value*

Simply prints out the type and value of *value*. This is mostly to refresh one's memory as to the values of symbolic names, though it can also be used as a simple calculator.

Restrictions: none.

`quit`

Exit **Ghu** and return to the system command level.

Restrictions: allowed/not-allowed (enables us to have an account which may *only* run **Ghu**).

`input` *filename*

Read **Ghu** commands from the indicated file, returning to terminal input upon reaching the end of the file.

Restrictions: only those on the input files themselves.

General Database Operations

`contents` *obspec-r*

Prints a list of the entities contained by the entity specified by *obspec* (a region by default, though it may also be an object or an Avatar). Each entity is listed on a line with its global ID and the values of its fundamental properties (x-y position, orientation, etc.).

Restrictions: *obspec* must be readable according to the global restrictions.

`find` *obspec-o*

Print out the chain of containership from the entity *obspec* (an object by default, though it may also be an Avatar) to the outermost containing region, listing the global ID and class of each entity in the chain.

Restrictions: *obspec* must be readable according to the global restrictions.

`get` *obspec-o*

Makes the entity indicated by *obspec* the ''working'' entity. By default it is an object, though it may also be a region or an Avatar.

Restrictions: *obspec* must be readable according to the global restrictions.

`display` [*obspec-o*] [`.` *field*]

Makes the entity indicated by *obspec* (an object by default, though it may also be a region or an Avatar) the ''working'' entity and then prints out the values of its fields. If *field* is given, it only prints the value of that particular field. If *obspec* is omitted, it defaults to the current ''working'' entity.

Restrictions: *obspec*, if given, must be readable according to the global restrictions.

`set` [*obspec-o* `.`] *field* [`=`] *value* [`!`]

Set the value of the property named by *field* in the ''working'' entity to the value specified by *value*. *value* must have a type that matches **Ghu**'s internal table definition for *field*. If *field* is preceded by *obspec*., the entity specified by *obspec* (an object by default, though it may also be a region or an Avatar) is first made the ''working'' entity. If the command is terminated with an exclamation point (''`!`''), write the new version of the entity to the database as if an `update` command had been issued.

Restrictions: *obspec*, if given, must be readable according to the global restrictions. It must also be writable if the `!` option is used.

`update` [*obspec-o*]

Write the ''working'' entity out to its associated database. If the entity is in an active region and has been changed since the last read from the database, broadcast **`FIDDLE`** messages(s) to all the players in the region informing them of the change(s). If *obspec* is explicitly specified, write the ''working'' entity to *obspec*'s database record rather than to it's own. If you do this, they must be the same type of entity and, if objects, of the same class; furthermore, certain fields of Avatar and region records are never changed by a write such as this.

Restrictions: the record must be writable according to the global restrictions.

`add` [*type-o*] [*class* [*value* [`,` *value*]*]]

Create a new entity of the given type (an object by default). One can specify its class (if it is an object) and the values of its property fields (in order) on the command line (any that are omitted will be prompted for). The entity becomes the ''working'' entity. If the entity is placed into an active region, broadcast a **`HEREIS`** request to all the players in the region informing them of the new item.

Restrictions: separate allowed/not-allowed flags for the creation of regions, objects, and Avatars. In the case of the objects and Avatars, the containing region and/or object must be writable according to the global restrictions. In the case of objects, there is a further per-class allowed/not-allowed flag.

`remove` *obspec-o*

Effectively delete the specified entity (an object by default, though it may also be an Avatar) from the database by moving it to a "limbo" region. If the entity is an Avatar its associated player must not be online and the player's user record should be updated to indicate that he or she no longer has an Avatar in the Habitat universe. If the entity is in an active region, broadcast a **`GOAWAY`** request to all the players in the region informing them of the disappearance.

Restrictions: separate allowed/not-allowed flags for the removal of objects and Avatars. Furthermore, *obspec* and its containing objects and regions must be writable according to the global restrictions.

Specialized Database Operations

`list` ⟦ *regionid1* ⟧ ⟦ `--` ⟧ ⟦ *regionid2* ⟧ ⟦ `active` ⟧

Print a list of the regions in the range from *regionid1* to *regionid2* (inclusive). The *regionid* values must have type `r` or `u`. If *regionid1* is omitted, it defaults to 0, i.e., the beginning of the database. If *regionid2* is omitted, it defaults to **MAXINT**, i.e., the end of the database. If only one *regionid* is given and the `--` is omitted, only the one region specified is listed. If both *regionid*s are omitted the `--` may be omitted as well. Omitting both means that the entire region database is to be listed. If the keyword `active` is given, only regions that are currently active (i.e., memory resident) are listed. For each region, the `list` command prints a line containing the region ID number, those of its four neighbors (-1 if there is no neighbor in a particular direction), the region's orientation, and the number of the regionproc with which the region is associated (if it is active). For example,

`list 5`	lists region number 5
`list 5--10`	lists regions 5 through 10
`list 5--`	lists from region 5 to the end of the database
`list --10`	lists from the start of the database to region 10
`list --`	lists all regions
`list`	also lists all regions
`list active`	lists all active regions

`nuke` ⟦ *regionid1* ⟧ ⟦ `--` *regionid2* ⟧

Removes the specified regions from the database. The regions are specified as in the `list` command, except that the ranges of regions must have explicit start and end numbers. It is not possible to `nuke` active regions.

Restrictions: there is a restriction list for this command similar to the global restriction lists. For most users, this list will be empty.

`kill` *obspec-a*

`kill` is similar to remove, but designed for online Avatars. Kill the specified Avatar (i.e., start it going through the normal Avatar death-and-reincarnation cycle), sending the appropriate messages to the Avatar and to the other players in its region as if it been killed by one of them.

Restrictions: *obspec* must be writable according to the global restrictions.

`read` ⟦ *obspec-o* ⟧

Prints out the text of the indicated document. If *obspec* has type `o` it must refer to a paper or book object, and that object's text entry is printed. If *obspec* has type `u` it is used directly as the text ID number.

Restrictions: allowed/not-allowed flags for the portions of the text ID space corresponding to books and paper respectively (separation is so **Ghu** users will not generally be able to read player mail).

`text` *obspec-u* ⟦ ⟦`<`⟧ *filename* ⟧

If *obspec* has type `u`, it is interpreted as a text ID. Otherwise, it should represent a book or paper object from which the text ID field will be extracted. Associate new text with the text ID in the text database. If *filename* is given, take the new text from the indicated file, otherwise prompt for it. There is a limit of 16 lines per page with no more than 40 characters per line allowed. Page breaks are indicated by lines consisting solely of form-feed characters (the form-feed lines do not go into the text database and do not count towards the 16 line per page limit).

Restrictions: allowed/not-allowed flag. If *obspec* is a paper or book object, then it must be writable according to the global restrictions. For ordinary text IDs there is a restriction list similar to the global restriction lists that constrains on the basis of text ID number.

`teleport` *address* [[`=`] *obspec-o*]

Print the global ID of the teleport booth associated with the teleport address *address* (a string value). If *obspec* is given, it must correspond to a teleport or elevator object; make the entry in the teleport address database for *address* point to the teleport indicated *obspec*.

Restrictions: allowed/not-allowed flags for read and write.

Online Operations

`players`

Print a list of the currently online players in the Habitat, together with appropriate information (user names, global ID's, location, etc.).

Restrictions: allowed/not-allowed flag.

`voice` [*obspec-r*] [*text*]

Broadcast to the indicated entity (a region by default) a **`SPEAK`** message from God containing the text *text* (a string value). If *text* is omitted it will be prompted for. The *text* must be short enought to fit in a single packet. If *obspec* is omitted and some entity is currently being `watch`'ed (see below), the text is sent to that entity. Otherwise an error results.

Restrictions: allowed/not-allowed flag. *obspec* must be writable according to the global restrictions.

`watch` [*obspec-r* | `off`]

If an *obspec* is given (a region by default, though it may also be an Avatar), monitor message traffic associated with the indicated entity. If *obspec* is a region, display all requests sent to or from that region. If *obspec* is an Avatar, display all messages sent by or to the associated player. *obspec* may not be an ordinary object. In all cases, **Ghu** *must* place a ghost icon in the region with the players affected, to inform them that the eavesdropping is taking place.

Once a `watch` command is issued, one may continue issuing other commands. The messages will simply be printed as they arrive. Entering another `watch` command supercedes the previous one (it is possible to `watch` only one entity at a time). Giving the keyword `off` in place of the *obspec* turns `watch`'ing off (and removes the ghost icon).

Restrictions: allowed/not-allowed flag. *obspec* must be readable according to the global restrictions.

`oracle` [*obspec-o* | `off` | `old`]

Attach the operator to the given oracle. *obspec* must refer to an oracular object. Henceforth any messages sent to that oracle will be printed along with their player of origin. Entering another `oracle` command supercedes the previous one (is is possible to listen to only one oracle at a time). `oracle` is a special form of `watch`, and `watch` commands and `oracle` commands also supercede each other. The object indicated in an `oracle` command becomes the `watch`ed entity for purposes of sending `voice` messages, etc. Entering the keyword `off` instead of an *obspec* turns oracle monitoring off. Entering the keyword `old` instead of an *obspec* yields a printout of old, unattended messages to that oracle that have

been written to the oracle log file.

Restrictions: allowed/not-allowed flag. There is additionally a restriction list (by oracle object ID number) similar to the global lists.

`mail` *username* [[<] *filename*]

Send a Habitat mail letter to the indicated user. **Ghu** will prompt for the text. There is a limit of 16 lines of no more than 40 characters per line. If *filename* is given, the text of the mail message is taken from the file named.

Restrictions: allowed/not-allowed flag.

`processes`

Print a list of the running Habitat processes associated with the incarnation of the Habitat universe that this invocation of **Ghu** is attached to. For region processes, list the regions that those processes are handling and the number of ghosts and Avatars currently in each. For other kinds of processes, print whatever other information may prove useful.

Restrictions: allowed/not-allowed flag.

The Ghu Command Line

The **Ghu** command issued from VOS command level on the Stratus has the following form:

`ghu` [*databasedir*] [`-do` *commandstring*]

where *databasedir* is the pathname of a directory in which the various database files for a running Habitat system will be found. If omitted it defaults to whatever the directory containing the current operational system is. *commandstring* is an optional string of **Ghu** commands that will be parsed like an ordinary **Ghu** command line. If the `-do` option is used, the commands in *commandstring* will be executed and then **Ghu** will exit. If `-do` is not given, **Ghu** will come up in interactive mode and start reading command lines from the terminal until a `quit` command or an end-of-file is encountered.