Chapter 5. Text User Interface

TGrid user interface also consists of a textual command line reference. The text user interface (TUI) is written in a dialect of Lisp called Scheme. Users familiar with Scheme will be able to use the interpretive capabilities of the interface to create customized commands. The TUI is described in the following sections:

- Section 5.1: Text Menu System
- Section 5.2: Text Prompt System
- Section 5.3: Interrupts
- Section 5.4: System Commands
- Section 5.5: Text Menu Input from Character Strings
- Section 5.6: Using the Text Interface Help System

5.1 Text Menu System

The text menu system provides a hierarchical interface to the program's underlying procedural interface.

- You can easily manipulate its operation with standard text-based tools: input can be saved in files, modified with text editors, and read back in to be executed, because it is text based.
- The text menu system is tightly integrated with the Scheme extension language, so it can easily be programmed to provide sophisticated control and customized functionality.

The menu system structure is similar to the directory tree structure of UNIX operating systems. When you first start TGrid, you are in the "root" menu and the menu prompt is simply a caret:

```
>
```

To generate a listing of the submenus and commands in the current menu, press `<RETURN>`.

```
>
file/             mesh/         report/
boundary/         display/      exit
```

By convention, submenu names end with a `/` to differentiate them from menu commands. To execute a command, type its name (or an abbreviation). Similarly, to move down into a submenu, enter its name or an abbreviation. When you move into the submenu, the prompt will change to reflect the current menu name.

```
> display

/display> set

/display/set>
```

To move back to the previously occupied menu, type `q` or `quit` at the prompt.

```
/display/set> q

/display>
```

You can move directly to a menu by giving its full pathname.

```
/display> /file

/display//file>
```

In the above example, control was passed from `/display` to `/file` without stopping in the root menu. Therefore, when you quit from the `/file` menu, control will be passed directly back to `/display`.

```
/display//file> q

/display>
```

If you execute a command without stopping in any of the menus along the way, control will again be returned to the menu from which you invoked the command.

```
/display> /file start-journal jrnl
 Input journal opened on file "jrnl".

/display>
```

The text menu system provides on-line help for menu commands. The text menu on-line help system is described in Section 5.6: Using the Text Interface Help System.

5.1.1 Command Abbreviation

To select a menu command, you need not type the entire name; you can type an abbreviation that matches the command.

- A command name consists of “phrases” separated by hyphens.
- A command is matched by matching an initial sequence of its phrases.
- Matching of hyphens is optional.
- A phrase is matched by matching an initial sequence of its characters.
- A character is matched by typing that character.

The rules for “matching” a command are:

- If an abbreviation matches more than one command, then the command with the greatest number of matched phrases is chosen.
- If more than one command has the same number of matched phrases, then the first command to appear in the menu is chosen.

  For example, each of the following will match the given command `set-ambient-color`: `set-ambient-color`, `s-a-c`, `sac`, and `sa`.
- When abbreviating commands, sometimes your abbreviation will match more than one command. In such cases, the first command is selected.
- Occasionally, there is an anomaly such as `lint` not matching `lighting-interpolation` because the `li` gets absorbed in `lights-on?` and then the `nt` doesn't match `interpolation`.

  This can be resolved by choosing a different abbreviation, such as `liin`, or `l-int`.

5.1.2 Scheme Evaluation

If you enter an open parenthesis "(" at the menu prompt, then that parenthesis and all characters up to and including the matching closing parenthesis are passed to Scheme to be evaluated. The result of evaluating the expression is then displayed.

```
> (define a 1)
a

> (+ a 2 3 4)
10
```

5.1.3 Aliases

Command aliases can be defined within the menu system. As with the UNIX `csh` shell, aliases take precedence over command execution. The following aliases are predefined in Cortex (`error`, `pwd`, `chdir`, `ls`, `.`, and `alias`):

`error` displays the Scheme object that was the "irritant" in the most recent Scheme error interrupt.

`pwd` prints the working directory in which all file operations will take place.

`chdir` changes the working directory.

`ls` lists the files in the working directory.

`alias` displays the list of symbols currently aliased.

5.2 Text Prompt System

Commands require various arguments, including numbers, filenames, yes/no responses, character strings, and lists. A uniform interface to this input is provided by the text prompt system. A prompt consists of a prompt string, followed by an optional units string enclosed in parentheses, followed by a default value enclosed in square brackets

```
filled grids? [no]  <RETURN>

shrink-factor [0.1]  <RETURN>

line-weight [1]  <RETURN>

title [""]  <RETURN>
```

- The default value for a prompt is accepted by typing a `<RETURN>` or a comma (`,`). A comma is not a separator. It is a separate token that indicates a default value. The sequence “`1,2`” results in three values
  - the number 1 for the first prompt.
  - the default value for the second prompt.
  - the number 2 for the third prompt.
- A short help message can be displayed at any prompt by entering a `?`. (See Section 5.6: Using the Text Interface Help System.)
- To abort a prompt sequence, press `<Control-C>`.

5.2.1 Numbers

The most common prompt type is a number. Numbers can be either integers or real numbers. Valid numbers are, for example, `16`, `-2.4`, `.9e5`, and `+1e-5`.

- Integers can also be specified in binary, octal, and hexadecimal form.
- The decimal integer 31 can be entered as `31`, `#b11111`, `#o37`, or `#x1f`.
- In Scheme, integers are a subset of reals, so you do not need a decimal point to indicate that a number is real; `2` is just as much a real as `2.0`.
- If you enter a real number at an integer prompt, any fractional part will be truncated. That is, `1.9` will become `1`.

5.2.2 Booleans

Some prompts require a yes-or-no response. A yes/no prompt will accept either `yes` or `y` for a positive response, or `no` or `n` for a negative response. These prompts are used for confirming potentially dangerous actions such as overwriting an existing file, exiting without saving case, data, mesh, etc. Some prompts require actual Scheme boolean values (true or false). These are entered with the Scheme symbols for true `#t` and false `#f`.

5.2.3 Strings

Character strings are entered in double quotes, as in `"red"`. Plot titles and plot legend titles are examples of character strings. Character strings can include any characters, including blank spaces and punctuation.

5.2.4 Symbols

Symbols are entered *without* quotes. Zone names, surface names, and material names are examples of symbols. Symbols must start with an alphabetical character (i.e., a letter), and cannot include any blank spaces or commas.

You can use wild cards (*) to specify zone names when using the TUI. Some examples are:

- `/display/boundary-grid *` allows you to display all the boundary zones in the mesh.
- `/boundary/delete-island-faces wrap*` allows you to delete island faces on all wrapper zones prefixed by `wrap`.

If you use a wild card for an operation that requires a single zone as input, you will be prompted to specify a single zone from the list of those that match the expression specified.

```
> /boundary/manage/name wall*  <RETURN>

wall-1     wall-2     wall-3
wall-4     wall-5     wall-6

Zone Name [ ]
```

5.2.5 Filenames

Filenames are actually character strings. For convenience, filename prompts do not require the string to be surrounded with double quotes. If, for some exceptional reason, a filename contains an embedded space character, then the name must be surrounded with double quotes.

One consequence of this convenience is that filename prompts do not evaluate the response. For example, the following sequence will end up writing a hardcopy file with the name `fn`, not `valve.ps`.

```
> (define fn "valve.ps")
fn

> hc fn
```

Since the filename prompt did not evaluate the response, `fn` did not get a chance to evaluate `"valve.ps"` as it would for most other prompts.

5.2.6 Lists

Some functions in TGrid require a "list" of objects such as numbers, strings, booleans, etc. A list is a Scheme object that is a sequence of objects terminated by the empty list, `'()`. Lists are prompted for an element at a time, and the end of the list is signaled by entering an empty list. This terminating list forms the tail of the prompted list, and can either be empty or can contain values.

For convenience, the empty list can be entered as `()` as well as the standard form `'()`. Normally, list prompts save the previous argument list as the default. To modify the list, overwrite the desired elements and terminate the process with an empty list.

For example the following creates a list of three numbers: 1, 10, and 100.

```
element(1) [()] 1
element(2) [()] 10
element(3) [()] 100
element(4) [()] <RETURN>
```

Subsequently, it adds a fourth element as shown below:

```
element(1) [1]  <RETURN>
element(2) [10]  <RETURN>
element(3) [100]  <RETURN>
element(4) [()]  <RETURN>
element(5) [()]  <RETURN>
```

Then it leaves only 1 and 10 in the list:

```
element(1) [1]  <RETURN>
element(2) [10]  <RETURN>
element(3) [100] ()
```

Subsequently entering creates a five element list: 1, 10, 11, 12, and 13.

```
element(1) [1] ,,'(11 12 13)
```

Finally, a single empty list removes all elements:

```
element(1) [1] ()
```

5.2.7 Evaluation

All responses to prompts (except filenames), are "evaluated" by the Scheme interpreter before they are used. Therefore you can enter any valid Scheme expression as the response to a prompt. For example, to enter a unit vector with one component equal to 1/3 (without using your calculator), do the following:

```
/foo> set-xy
x-component [1.0] (/ 1 3)
y-component [0.0] (sqrt (/ 8 9))
```

You can also first define a utility function to compute the second component of a unit vector as follows:

```
> (define (unit-y x) (sqrt (- 1.0 (* x x))))
unit-y
/foo> set-xy
x-component [1.0] (/ 1 3)
y-component [0.0] (unit-y (/ 1 3))
```

5.2.8 Default Value Binding

The default value at any prompt is bound to the Scheme symbol "_" (underscore) so that the default value can form part of a Scheme expression. For example, if you want to decrease a default value by one-third, you could enter

```
shrink-factor [0.8] (/ _ 3)
```

5.3 Interrupts

The execution of the code can be halted using `<Control-C>`, at which time the present operation stops at the next recoverable location.

5.4 System Commands

If you are running TGrid in a UNIX operating system, you can execute system commands with the ! (bang) shell escape character. All characters following the ! up to the next newline character will be executed in a subshell. Any further input related to these system commands must be entered in the window in which you started the program, and any screen output will also appear in that window.

If you start TGrid remotely, this input and output will be in the window in which you started Cortex.

```
> !rm junk.*

> !vi script.rp
```

The `ls` and `pwd` aliases invoke the UNIX `ls` and `pwd` commands in the working directory. The `chdir` alias changes the current working directory of the program. The `!ls` and `!pwd` commands will execute the UNIX commands in the directory in which Cortex was started. The screen output will appear in the window in which you started TGrid, unless you started it remotely, in which case the output will appear in the window in which you started Cortex.

Note: *Commands `!chdir` or `!cd` executes in a subshell, so it will not change the working directory either for* TGrid *or for* Cortex*, and it is therefore not useful.*

Typing `chdir` with no arguments will move you to your home directory in the console. Examples of system commands entered in the console are as follows:

```
> !pwd

> !ls valve.*
```

The screen output will appear in the window in which TGrid was started. If you start the program remotely, it will appear in the window in which Cortex was started.

```
/home/cfd/run/valve
valve1.cas   valve1.msh   valve2.cas   valve2.msh
```

5.5 Text Menu Input from Character Strings

When writing a Scheme extension function for TGrid, it is convenient to be able to include menu commands in the function. This can be done with `ti-menu-load-string`. For example, to open graphics window 1, use:

```
(ti-menu-load-string "di ow 1")
```

A Scheme loop that will open windows 0 and 1 and display the front view of the grid in window 0 and the back view in window 1 is given by

```
(for-each
 (lambda (window view)
   (ti-menu-load-string (format #f "di ow ~a gr view rv ~a"
window view)))
 '(0 1)
 '(front back))
```

This loop makes use of the `format` function to construct the string used by `menu-load-string`. This simple loop can also be written without using menu commands, but you need to know the Scheme functions that get executed by the menu commands to do it:

```
(for-each
 (lambda (window view)
   (cx-open-window window)
   (draw-mesh)
   (cx-restore-view view))
 '(0 1) '(front back))
```

String input can also provide an easy way to create aliases within TGrid. For example, to create an alias that will display the grid, you can type the following:

```
(alias 'dg (lambda () (ti-menu-load-string "/di gr")))
```

Then any time you enter `dg` from anywhere in the menu hierarchy, the grid will be drawn in the active window.

`ti-menu-load-string` evaluates the string argument in the top level menu. It ignores any menu you may be in when you invoke `ti-menu-loadstring`. Therefore, the command

```
(ti-menu-load-string "open-window 1 gr")    ; incorrect usage
```

will not work even if you type it from within the `display/` menu. The string itself must cause control to enter the `display/` menu, as in

```
(ti-menu-load-string "display open-window 1 grid")
```

5.6 Using the Text Interface Help System

The text user interface provides context-sensitive on-line help. Within the text menu system, a brief description of each of the commands can be invoked by entering a `?` followed by the command in question.

Example:

```
> ?dis
display/: Enter the display menu.
```

You can also enter a lone `?` to enter "help mode." In this mode, you need only enter the command or menu name to display the help message. To exit help mode type `q` or `quit` as for a normal menu.

Example:

```
> ?

[help-mode]> di
display/: Enter the display menu.

[help-mode]> pwd
pwd: #[alias]
(LAMBDA ()
  (cx-send '(system "pwd")))

[help-mode]> q

>
```

Help can also be obtained when you are prompted for information by typing a ? at the prompt.

Example:

```
> display/annotate
Annotation text [""] ?
Enter the text to annotate the plot with.
Annotation text [""]
```