32. Querying the User

The following functions provide a convenient and consistent interface for asking questions of the user. Questions are printed and the answers are read on the stream ***query-io***, which normally is synonymous with ***terminal-io*** but can be rebound to another stream for special applications.

The macro **with-timeout** (see page 686) can be used with the functions in this chapter to assume an answer if the user does not respond in a fixed period of time.

We first describe two simple functions for yes-or-no questions, then the more general function on which all querying is built.

y-or-n-p &optional *format-string* &rest *format-args*

This is used for asking the user a question whose answer is either 'y' for 'yes' or 'n' for 'no'. It prints a message by passing *format-string* and *format-args* to **format**, reads a one-character answer, echoes it as 'Yes' or 'No', and returns **t** if the answer is 'yes' or **nil** if the answer is 'no'. The characters which mean 'yes' are 'Y', 'T', **Space**, and **Hand-up**. The characters which mean "no" are 'N', **Rubout**, and **Hand-down**. If any other character is typed, the function beeps and demands a 'Y or N' answer.

You should include a question mark and a space at the end of the message. **y-or-n-p** does type '(Y or N)' for you.

query-io is used for all input and output.

y-or-n-p should be used only for questions that the user knows are coming. If the user is not going to be anticipating the question (e.g. if the question is "Do you really want to delete all of your files?" out of the blue) then **y-or-n-p** should not be used, because the user might type ahead a 'T', 'Y', 'N', **Space**, or **Rubout**, and therefore accidentally answer the question. In such cases, use **yes-or-no-p**.

yes-or-no-p &optional *format-string* &rest *format-args*

This is used for asking the user a question whose answer is either 'yes' or 'no'. It prints a message by passing *format-string* and *format-args* to **format**, beeps, and reads in a line from ***query-io***. If the line is 'yes', it returns **t**. If the line is 'no', it returns **nil**. (Case is ignored, as are leading and trailing spaces and tabs.) If the input line is anything else, **yes-or-no-p** beeps and demands a 'yes or no' answer.

You should include a question mark and a space at the end of the message. **yes-or-no-p** does type '(Yes or No)' for you.

query-io is used for all input and output.

To allow the user to answer a yes-or-no question with a single character, use **y-or-n-p**. **yes-or-no-p** should be used for unanticipated or momentous questions; this is why it beeps and why it requires several keystrokes to answer it.

fquery *options format-string* &rest *format-args*

Asks a question, printed by (format *query-io* *format-string format-args...*), and returns the answer. fquery takes care of checking for valid answers, reprinting the question when the user clears the screen, giving help, and so forth.

options is a list of alternating keywords and values, used to select among a variety of features. Most callers pass a constant list as the *options* (rather than consing up a list whose contents varies). The keywords allowed are:

:type — What type of answer is expected. The currently-defined types are :tyi (a single character), :readline or :mini-buffer-or-readline (a line terminated by a carriage return). :tyi is the default. :mini-buffer-or-readline is nearly the same as :readline, the only difference being that the former uses a minibuffer if used inside the editor.

:choices — Defines the allowed answers. The allowed forms of choices are complicated and explained below. The default is the same set of choices as the y-or-n-p function (see above). Note that the :type and :choices options should be consistent with each other.

:list-choices — If t, the allowed choices are listed (in parentheses) after the question. The default is t; supplying nil causes the choices not to be listed unless the user tries to give an answer which is not one of the allowed choices.

:help-function — Specifies a function to be called if the user hits the Help key. The default help-function simply lists the available choices. Specifying nil disables special treatment of Help. Specifying a function of three arguments—the stream, the list of choices, and the type-function—allows smarter help processing. The type-function is the internal form of the :type option and can usually be ignored.

:condition — If non-nil, a signal name (see page 713) to be signaled before asking the question. A condition handler may handle the condition, specifying an answer for fquery to return, in which case the user is not asked. The details are given below. The default signal name is :fquery, which signals condition name :fquery.

:fresh-line — If t, *query-io* is advanced to a fresh line before asking the question. If nil, the question is printed wherever the cursor was left by previous typeout. The default is t.

:beep — If t, fquery beeps to attract the user's attention to the question. The default is nil, which means not to beep unless the user tries to give an answer which is not one of the allowed choices.

:stream — The value should be either an I/O stream or a symbol or expression that will evaluate to one. fquery uses the specified stream instead of *query-io* for all its input and output.

:clear-input — If t, fquery throws away type-ahead before reading the user's response to the question. Use this for unexpected questions. The default is nil, which means not to throw away type-ahead unless the user tries to give an answer which is not one of the allowed choices. In that case, type-ahead

is discarded since the user probably wasn't expecting the question.

:make-complete

If t and *query-io* is a typeout-window, the window is "made complete" after the question has been answered. This tells the system that the contents of the window are no longer useful. Refer to the window system documentation for further explanation. The default is t.

The argument to the :choices option is a list each of whose elements is a *choice*. The cdr of a choice is a list of the user inputs which correspond to that choice. These should be characters for :type :tyi or strings for :type :readline. The car of a choice is either a symbol which fquery should return if the user answers with that choice, or a list whose first element is such a symbol and whose second element is the string to be echoed when the user selects the choice. In the former case nothing is echoed. In most cases :type :readline would use the first format, since the user's input has already been echoed, and :type :tyi would use the second format, since the input has not been echoed and furthermore is a single character, which would not be mnemonic to see on the display.

A choice can also be the symbol :any. If used, it must be the last choice. It means that any input is allowed, and should simply be returned as a string or character if it does not match any of the other choices.

Perhaps this can be clarified by example. The yes-or-no-p function uses this list of choices:

```
((t "Yes") (nil "No"))
```

and the y-or-n-p function uses this list:

```
(((t "Yes.") #\y #\t #\space #\hand-up)
 ((nil "No.") #\n #\rubout #\hand-down))
```

If a signal name is specified (or allowed to default to :fquery), before asking the question fquery will signal it. (See section 30.1, page 698 for information about conditions.) make-condition will receive, in addition to the signal name, all the arguments given to fquery, including the list of options, the format string, and all the format arguments. fquery provides one proceed type, :new-value, and if a condition handler proceeds, the argument it proceeds with is returned by fquery.

If you want to use the formatted output functions instead of format to produce the promting message, write

(fquery *options* (format:outfmt *exp-or-string exp-or-string* ...))

format:outfmt puts the output into a list of a string, which makes format print it exactly as is. There is no need to supply additional arguments to the fquery unless it signals a condition. In that case the arguments might be passed so that the condition handler can see them. The condition handler will receive a list containing one string, the message, as its third argument instead of just a string. If this argument is passed along to format, all the right things happen.

fquery (condition) *Condition*

This condition is signaled, by default, by fquery. The condition instance supports these operations:

:options Returns the list of options given to fquery.

:format-string Returns the format string given to fquery.

:format-args Returns the list of additional args for format, given to fquery.

One proceed type is provided, :new-value. It should be used with a single argument, which will be returned by fquery in lieue of asking the user.

format:y-or-n-p-options *Constant*

A suitable list to pass as the first argument to fquery to make it behave like y-or-n-p.

format:yes-or-no-p-options *Constant*

A suitable list to pass as the first argument to fquery to make it behave like yes-or-no-p.

format:y-or-n-p-choices *Constant*

A list which y-or-n-p uses as the value of the :choices option.