The research reported in this paper was sponsored by the Advanced Research Projects Agency Information Processing Techniques Office and was monitored by the Electronic Systems Division, Air Force Systems Command under contract F1962867C0004 with the System Development Corporation.

TECH MEMO *a working paper*

System Development Corporation / 2500 Colorado Avenue / Santa Monica, California 90406
Information International Inc. / 11161 Pico Boulevard / Los Angeles, California 90064

TM- 2710/240/01

AUTHOR [illegible]
C. Weissman, SDC

TECHNICAL Donna Firth
Donna Firth, SDC

RELEASE [illegible]
L. Hawkinson, III
[illegible]
C. Weissman, SDC
for J. I. Schwartz, SDC

DATE 9/16/66 PAGE 1 OF 29 PAGES

LISP 2 Project

Input-Output

ABSTRACT

This document specifies the philosophy and mechanics of LISP 2 input-output, which is designed for maximum on-line interaction under time-sharing. LISP 2 data files are described as the principal input-output mechanism. These files may be created, positioned, selected, read, written, and dismissed, dynamically, at run-time, for the complete spectrum of physical I-O devices available to the system. The user may also control the external horizontal and vertical page formatting of symbolic files.

The mechanics of Q-32 LISP 2 file management are also described herein by specification of a collection of I-O variables and primitives.

CONTENTS

CONTENTS (Cont'd)

FIGURES

1. INPUT-OUTPUT: FILES

LISP input-output is based upon a set of primitive functions and variables that is available to the user. Each input or output operation references, either implicitly or explicitly, a specific file. A file is associated with a particular device as declared by the user; the LISP meaning of the term "file" is the same as the meaning normally assigned by a time-sharing system. A file is device-dependent but direction-independent; the same file may be used for both input and output, and with caution the file can be used for both purposes simultaneously.

A file consists of a sequence of records. A record is stored as a LISP array of type "OCTAL" if the file is binary and as a string if the file is symbolic. To reduce buffer storage overhead, only one record for a given file can be in main memory at a time. A file can contain any number of records.

A symbolic record consists of a sequence of lines, possibly only one. The number of lines per record depends upon either the physical device, the user's requirements, or both. A line, in turn, consists of a sequence of characters. The number of characters per line may be specified by the user within the limits imposed by the physical device. If this number is not specified it is assumed to be 72.

A binary record consists of a sequence of machine words of some user-controlled number.

When a record is transmitted between a device and main memory, control characters are usually required to indicate line formatting. Since these characters may vary from device to device and machine to machine, LISP represents this information in the form of an integer array of control words associated with the file. When a record is transmitted from main memory to a device during output, the control words are used to generate appropriate control characters. The process is reversed during input. This editing process is called string post-processing. The control array includes provisions for the logical concept of a (printed) page. There is no necessary relationship between page boundaries and record boundaries, and the user can adjust page sizes to suit his format requirements. Page terminations are handled automatically in accordance with the information in the control array.

A binary record is composed of a sequence of words. If a word is taken to be a single line, file control for binary files can be made to behave like that for symbolic files. Externally, binary data records do not contain any control information.

1.1 FILE ACTIVATION AND SELECTION

A file may be either available or inactive; an available file, in turn, may be either selected or deselected. No record is kept within LISP of inactive files, and no space is allocated for them. At any given time, exactly one file is selected for input and one for output; all other available files are deselected. A single file can be selected for both input and output simultaneously. The reading functions all operate on the currently selected input file; the printing functions all operate on the currently selected output file.

Each available file has associated with it the following four arrays for which the user supplies the information:

(1) The data record string.

(2) An integer array of format control variables.

(3) A functional array of overflow and string post-processing variables.

(4) A symbol array of miscellaneous control data.

When a file is selected, various locations within these arrays are bound to a family of fluid locative variables. The reading or printing functions, as the case may be, then access the file arrays through the family of variables. When a file is deselected, the file arrays are frozen in their current state until the file is again selected. The state is preserved (unless explicitly modified) down to the level of the single character about to be read or printed.

Once a symbolic file is selected, the symbolic input-output primitives act only on that file. Thus it is possible to write a LISP 2 program that is independent of device, format, etc., by supplying the name of the file as an argument of the program. For instance, one can debug a program by supplying input from teletype that will ultimately come from tape; at the time of switchover, only the file name need be changed.

There are four functions that change the state of a file. Each of them reference a file name. These functions are:

(1) OPEN, which makes an inactive file available or creates a new available file.

(2) SHUT, which makes an available file inactive.

(3) INPUT, which selects an available input file and deselects the previous input file.

(4) OUTPUT, which selects an available output file and deselects the previous output file.

Each of these will now be discussed in detail.

1.1.1 OPEN Function

A file is created by evaluating the function OPEN. OPEN establishes all necessary communication linkages between LISP 2 and the time-sharing monitor, and accomplishes the following:

(1) Creates an internal string sufficient in size to contain one data record for the designated file.

(2) Creates an integer and a symbol array and sets their contents with format and control information for the file.

(3) Creates a functional array and sets its contents with necessary primitives for formatting and string post-processing.

(4) Declares information to the time-sharing monitor, enabling the monitor to establish communication linkages with the external storage medium designated for the file.

(5) Maintains a list of all available file names and their descriptions as the value of fluid variable FILES.; all active file names are derived from the value of fluid variable FILES. and returned as the value of OPEN. OPEN uses FILES. to check for redundant or conflicting file names.

(6) Stores the file's name and description as a dotted pair on the list bound to variable FILES. .

OPEN is a function of two arguments and has the form

```
OPEN(name,desc)
```

where name is the name of the file being created and desc is a description of the file.

The name of the file being created must be a LISP 2 identifier. The first six characters of the name will be used by Q-32 LISP 2 as the internal name for the file in establishing its communication linkages with the time-sharing monitor. Therefore, the first six characters must be unique among all previously opened files. For file names of less than six characters, OPEN will create a six-character name by filling the remaining character positions with blanks.

The description of a file is a list of flags and dotted pairs of attributes and variables, in property-list format. This list specifies the unit, form, connection, protection, identification, and format of the file, and is discussed in Section 1.2. Except for nonstandard input-output operations, the user need not concern himself with the construction of a file description list, since he can use an appropriate standard description. The standard descriptions are specified in Section 1.2.2.

1.1.2 SHUT Function

A file created by OPEN may be made inactive and purged from LISP 2 by evaluating the function SHUT. SHUT breaks all the communication linkages and deletes all internal structures such as arrays, strings, and variables that were dynamically established by OPEN. Before purging the file, SHUT communicates to the time-sharing monitor the disposition of the file, e.g., add the file to a permanent inventory, delete all file references, change the protection mode of the file, change the file name, etc. The value of SHUT is all the active file names derived from the value of the fluid variable FILES. .

SHUT is a function of two arguments, and has the form

    SHUT(name, disp)

name must be a LISP 2 identifier naming an available file, i.e., a file name previously used with OPEN. If name is not in the list FILES., then SHUT has no effect. disp is a list of dotted pairs of attributes and values specifying the disposition of the file.

SHUT does not cause any data to be moved out of the file; it is left to the user to call for an ENDOUTR and POSITION(WEOF.), when appropriate, before calling SHUT.

1.1.2.1 File Disposition List

The dotted pair

    (FILE . x)

in the file disposition list is used to communicate to the time-sharing monitor the inventory action desired for the file. The action is specified by the parameter x. For Q-32 LISP 2, x may be SAVE or DELETE. Saved files are inventoried and may be accessed at a later date by the use of the flag OLD (cf. Section 1.2.1.3) as the connection mode of an OPEN call. DELETEd files will not be inventoried and will disappear. If the CDR of the FILE pair is DELETE, no further disposition parameters are meaningful.

The dotted pair

    (NAME . file-name)

may be used to change the name of the saved file in the time-sharing system inventory. If this parameter is absent, the file name under which the file was opened will be used. In either case, SHUT may be forced to query the on-line user for a new file name if the file name (a name local to LISP) conflicts with a name already in the inventory.

The dotted pair

```
(PROTECT . y)
```

may be used to change the file protection of the saved file. If this parameter is absent, the protection mode under which the file was opened will be used (cf. Section 1.1.3.4). For Q-32 LISP 2, this parameter will be ignored.

The dotted pair

```
(LOG . z)
```

may be used to log a message on the operator's console. The parameter z is that message, in the form of a LISP 2 S-expression. LOG is useful for giving directions to console operators, labels to tapes, etc.

1.1.3 File Selection

The file selection calls are of the form

```
INPUT(name)
OUTPUT(name)
```

where name is the name of an active file, i.e., used in a previous OPEN call. INPUT is the function used for selecting input files for reading. OUTPUT is the function used for selecting output files for printing. The value of INPUT or OUTPUT is the file name previously selected. By using INPUT or OUTPUT as the second argument of an assignment expression, the name of the previously selected file can be saved for subsequent reselection, e.g.,

```
x←OUTPUT(name)
```

will save the name of the deselected file as the value of symbol variable x; subsequently, the expression

```
OUTPUT(x)
```

can be used to reselect the previously deselected file.

When a file is selected, the locations of the four arrays that specify the state of the file are retrieved. The fluid locative variables that are used by the reading and printing primitives are then bound to various locations within these arrays. Through this mechanism, one file can be deselected and another selected without any extraneous data copying, and yet the control variables for reading and printing can be accessed through a single point that is independent of which file is currently selected.

Selection and deselection have no effect on the state of a file (i.e., the state of its control parameters). Thus, deselection of files is permitted in the middle of lines and records. A partial line or record exists in terms of the state of the file's control parameters.

1.2 FILE DESCRIPTIONS

A file description specifies various relevant properties of the file. It is initialized at the time that a file is opened; some of the properties can be altered later (cf. Section 2). In Q-32 LISP, the changeable parameters are HORIZONTAL, VERTICAL, OVERFLOW, and PROTECT. The following properties may appear:

```
UNIT
FORM
PROTECT
REEL
SCOPE
NAME
RECORD
HORIZONTAL
VERTICAL
OVERFLOW
```

In addition, either or neither of the flags NEW and OLD may appear. All of these properties and flags will be discussed in the following subsections.

A property may appear more than once in a file description; in this case, the first appearance from the left is the one that counts. Thus, properties may be added at the front of a standard file description to modify that file description. For example, one might call

```
OPEN('TAPE01, '(REEL . 1234) . TAPE.)
```

to open file TAPE01 with reel 1234 and the standard tape file format as described by the system variable TAPE. (cf. Section 1.2.2).

1.2.1 Entries in the File Description

1.2.1.1 Unit

A file is unit-dependent, because LISP 2 uses the unit type of file to establish the proper communication linkages with the time-sharing monitor, and to set up the correct string post-processing for the file. Thus one element of the file description list must be a dotted pair designating the unit type. For example, the dotted pair

```
(UNIT . TTY)
```

designates the on-line typewriter or teletype as the unit for a particular file by the presence of the identifier TTY as the CDR of the dotted pair. Similarly,

```
(UNIT . DISC)
(UNIT . TAPE)
(UNIT . CORE)
(UNIT . CRT)
```

will designate disc, tape, core, and CRT (scope display), respectively, as the unit for a file.

1.2.1.2 Form

Symbolic data will be represented internally in LISP 2 as 8-bit ASCII characters. However, not all external media use this standard, and conversion will be required. These conversions will be performed by primitives as part of string post-processing.

At least two types of symbolic converters will be required for Q-32 LISP 2. The converter is specified by the value of the attribute FORM in the file description list. The value may be ASCII, BCD, or BINARY.

Q-32 teletypes use a 12-bit representation of ASCII 8-bit code. The dotted pair

```
(FORM . ASCII)
```

will be used to invoke a 12-bit to 8-bit (for teletype input) or an 8-bit to 12-bit (for teletype output) converter. ASCII may also be used with tape and disc with appropriate conversion performed.

```
(FORM . BCD)
```

will be used to invoke a 6-bit to 8-bit, or 8-bit to 6-bit converter.

```
(FORM . BINARY)
```

specifies a binary file, and no conversion is necessary, because a simple binary record will be transferred to and from the file. When a binary file is selected, the primitive functions READWORD and PRINWORD must be used, rather than READ and PRINT.

1.2.1.3 Connection

When OPEN establishes communication linkages with the time-sharing monitor, LISP must tell the monitor how to connect the external file to its internal storage area. LISP is made aware of whether or not the file already exists in the file inventory of the time-sharing monitor by means of a <u>connection flag</u>, which is one of the identifiers "NEW" or "OLD." The user is responsible for specifying the connection flag which is part of the file description list; if it does not appear, NEW is assumed. If the file is an old file, the monitor will connect

LISP to the existing copy of it; if the file is a new file, the monitor will allocate storage for the file on the requested external device and then connect LISP to the device.

1.2.1.4 Protection

File security is a difficult problem in time-sharing systems, and data files must be protected by the monitor from inadvertent or malicious acquisition by unauthorized persons. In LISP 2, the presence of the dotted pair

(PROTECT . x)

in the file description list is used to convey "keys" to lock or unlock various protected files. The nature of the parameter x depends upon the protection schemes provided by the monitor.

For Q-32 LISP 2, all files are assumed to be private to the current user. Also an interpretive protection scheme will be provided with Q-32 LISP 2. The parameter x is a list that may contain the identifiers READ or WRITE, or both. If READ is present, the file will be READ-protected (i.e., reading is prohibited). If WRITE is present, writing (printing) is prohibited. The prohibition mechanism works through the functions INPUT and OUTPUT already discussed. The functions will balk if a protected file is selected for input or output. As a further protection for tape files, the "ring" will be removed from the physical tape for WRITE-protected tape files. For other LISP 2 implementations, the parameter x can designate a password, a protection code, an executable protection function, a change of protection code, or a combination of these.

1.2.1.5 Identification

This parameter is optional, and used where it is desired to identify a specific physical unit. The dotted pair is of the form

(y . z)

where y and z take on the following values for Q-32 LISP 2.

Unit	y	z	Comment
TAPE	REEL	$n \leq 9999$	Physical reel number; if identification parameters absent, a default of reel n=0 (scratch tape) will be assumed.
CRT	SCOPE	$1 \leq n \leq 6$	Physical scope number.
DISC	NAME	identifier	Disc file name. If it is different from the first argument of OPEN, it will be used as the name of the file for communication with the time-sharing system.

For other LISP 2 implementation, y and z may take on other values and meanings.

1.2.1.6 Format

The last dimension of a file to be considered is its format. Format means the external organization of the file, particularly its blocked structure and its printed structure. Within the physical limitations of the hardware, the user may control these formats; if he does not, the system will set the necessary parameters by default. These dotted pairs, like the prior parameters, are top-level elements of the file description list.

1.2.1.6.1 Record Length. The format parameter RECORD specifies the number of lines to be blocked in each record. This parameter is permanent for the life of a file. The dotted pair is of the form

(RECORD . n)

where n is the integer number of lines. It is imperative that n be at least the number of lines in records being read from an external unit. That is, if a tape is written by LISP 2 (or any other system) as n lines per record, then it must be read with n lines per record, otherwise data can be lost. For Q-32 LISP 2, and for most time-sharing systems in general, it is desirable to read or write maximum-sized records for faster I/O. The following table specifies the upper bound on n for Q-32 LISP 2 units.

Unit	Form	n Max	n Default	Comment
TTY	ASCII	1	1	Record ≡ 1 line of 72 characters
TAPE	BCD	30	30	Line ≡ card image of 72 characters
	BCD	20		Line ≡ 120 characters
	ASCII	optional		Line ≡ optional
	BINARY	optional		Line ≡ binary word
DISC	BCD	50	50	Line ≡ card image of 72 characters plus 8 control characters
	ASCII	optional		Line ≡ optional
	BINARY	optional		Line ≡ binary word
CRT*		680 n	680	Line ≡ binary word; $1 \leq n \leq 3$
CORE*		optional	1	Default will consider a CORE file ≡ 1 record ≡ 1 line of optional character size

*These units are not currently available for Q-32.

1.2.1.6.2 Page Format. Page format is controlled by the three properties HORIZONTAL, VERTICAL, and OVERFLOW. The effects of these parameters are described in detail in Section 2.1.

Briefly, HORIZONTAL has three parameters associated with it: a left margin, a right margin, and a maximum column. These parameters in section IO are accessible for the currently selected file through the fluid variables LMG, RMG, and MAXCOL, respectively for output, and ILMG, IRMG, and IMAXCOL, respectively for input. The fluid variables are bound to the parameter values when the file is selected. The property is stored in the file description list in the form

```
(HORIZONTAL . (x y z))
```

where x, y, and z are integers specifying the left margin, right margin, and maximum column, respectively.

Similarly, VERTICAL has an upper line boundary, a lower line boundary, and a page boundary. For output, these are represented, in section IO, respectively by the fluid variables TOP, BOT, and PAGE, and for input by the fluid variables ITOP, IBOT, and IPAGE. The property is stored in the file description list in the form

```
(VERTICAL . (x y z))
```

where x, y, and z are integers specifying the upper line boundary, the lower line boundary, and the page boundary, respectively. If the column counter moves beyond a right-hand column boundary, or if the line counter moves below a lower boundary, an overflow condition results. Overflow can also result from exceeding a record boundary. Overflow actions for the record boundary, the maximum column and the page boundary are built into the system; those for the right margin and the lower line boundary can be specified by the user. These are specified by the property

```
(OVERFLOW . (x y))
```

where x and y are functional constants (of no arguments and NOVALUE) that specify the actions to be taken for right margin and lower line boundary overflow, respectively.

1.2.2 Reserved I/O Variables

There is a set of reserved variables whose values are file descriptions for various input-output devices. These file descriptions may be modified as described at the beginning of Section 1.2. The variables and their values are as follows:

```
(1)  TTY.     ((UNIT . TTY) (FORM . ASCII) (RECORD . 1)
               (HORIZONTAL . (1 73 72)))

(2)  DISC.    ((UNIT . DISC) (FORM . BCD) (RECORD . 50)
               (HORIZONTAL . (1 73 80)) (VERTICAL . (1 51 50)))

(3)  TAPE.    ((UNIT . TAPE) (FORM . BCD) (RECORD . 30)
               (HORIZONTAL . (1 73 80)) (VERTICAL . (1 51 50)))

(4)  CORE.    ((UNIT . CORE) (FORM . ASCII) (RECORD . 1))

(5)  CRT.     ((UNIT . CRT) (FORM . BINARY) (RECORD . 680))
```

CORE will eventually be used as an input-output unit in order to cause the printing and reading functions to generate or absorb internal strings. It is not implemented currently on Q-32 LISP 2.

2. INPUT-OUTPUT: FORMAT CONTROL, READING, PRINTING

The format of a printed page is specified at the time that a symbolic file is opened, and may be modified subsequently. The various format variables are specified in Section 2.1. The reading and printing functions are specified in Section 2.2, and methods of modifying and interrogating the format variables are described in subsequent sections. All input-output variables are FLUID, LOC and are in section IO.

2.1 PAGE CONTROL VARIABLES

The structure of a printed page is logically considered to have two directions: horizontal and vertical. The horizontal position is measured in columns and the vertical position is measured in lines. The printed page is shown schematically in Figure 1 and variables are shown for output only. Each output variable has an identical input variable counterpart, the name of which is prefixed with an "I." For example, CURCOL is an output variable and ICURCOL is the input counterpart.

At any given time, the variable CURCOL has an integer value specifying the current output column, and the variable CURLINE has an integer value specifying the current output line. If a single character is printed, it will appear in the column and line specified by CURCOL and CURLINE on the selected output file. (If a file is deselected and reselected later, the values of CURCOL and CURLINE at the time of reselection are the same as those at the time of deselection.) The corresponding variables for input are ICURCOL and ICURLINE; if a single character is read, it will be the character in the column and line specified by ICURCOL and ICURLINE.*

2.1.1 Horizontal Control Variables

2.1.1.1 Left Margin (LMG and ILMG)

The fluid variable LMG has as its value a positive integer specifying the left margin for output. When a new line is started because of termination of the previous line by the function ENDOUT, CURCOL is initialized to LMG. The character positions preceding LMG are filled with blanks. Text can be formatted dynamically by changing the value of LMG.

The input variable corresponding to LMG is ILMG. When a new line is started because of termination of the previous line by the function ENDIN, ICURCOL is initialized to ILMG. The characters preceding ILMG are ignored.

*Whereas changes to CURCOL (e.g., CURCOL←27) position current column controls, changes to CURLINE (e.g., CURLINE←12) do not position current line controls. CURLINE changes only affect vertical overflow controls. To skip or space n lines, call ENDOUT (or ENDIN) n times.

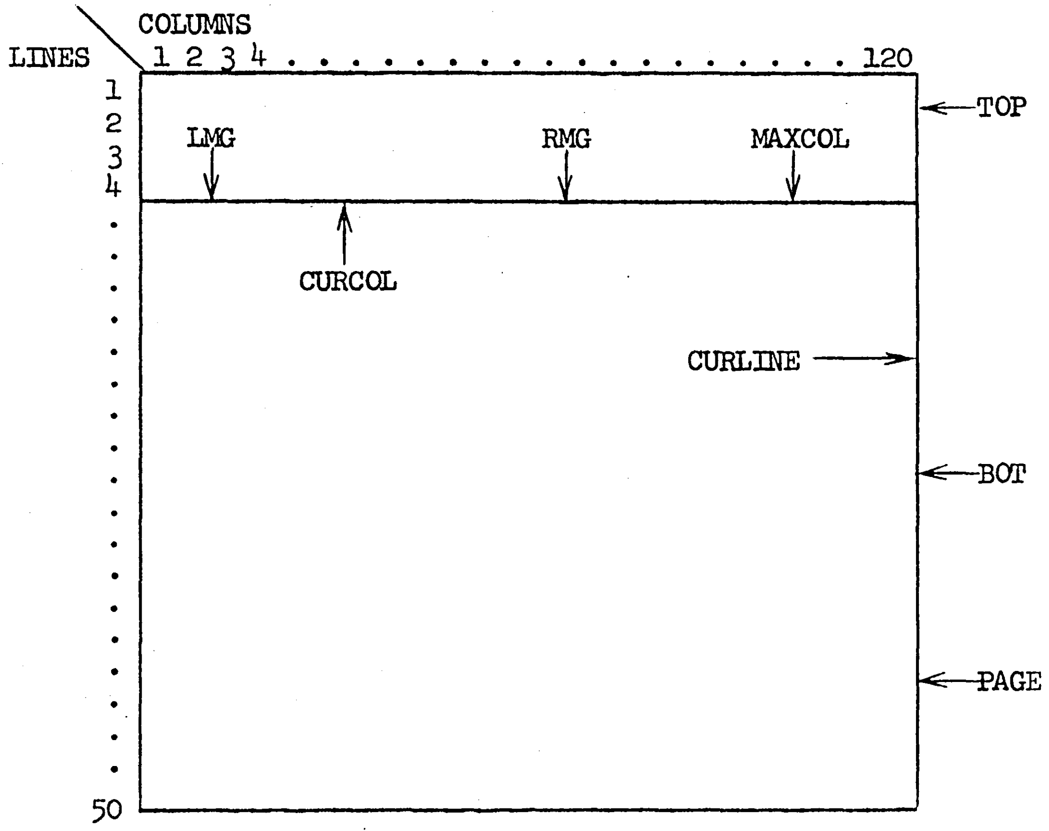

Figure 1. Page Format Controls Shown with Output Control Variables

2.1.1.2 Right Margin (RMG and IRMG)

RMG acts like the bell on a typewriter and warns the user that he is nearing the end of the output line. For each invocation of the character output function PRINCH (cf. Section 2.2.2), a check is made before the character is output to see if the condition CURCOL = RMG holds. If so, the RMG overflow function is invoked. RMG overflow is not invoked if CURCOL > RMG, so that if the right margin is bypassed, no RMG overflow will occur. The RMG overflow function is a functional constant of no arguments specified by the user or defaulted to ENDOUT by the system. Possible actions a user might take include inserting a hyphenator and ending the line. RMG may be greater than MAXCOL: in this case, RMG overflow will never occur.

The variable IRMG designates the right margin for the input file. IRMG overflow is invoked if, upon invoking the character reading function READCH (cf. Section 2.2.1), prior to the actual character read, the condition ICURCOL = IRMG holds. In all other respects the behavior of IRMG is exactly analogous to the behavior of RMG except that the system default functional constant is ENDIN.

2.1.1.3 Maximum Column (MAXCOL and IMAXCOL)

MAXCOL specifies the highest-numbered column in the output line. Before each application of PRINCH, a check is made to see if CURCOL > MAXCOL. If so, the line terminator ENDOUT (cf. Section 2.3.1) is invoked. MAXCOL must be constant over the life of a file and should never be changed by the user. The user can specify MAXCOL when a file is OPENed, otherwise it will default to 72 characters for teletype and 80 characters for tape and disc files on the Q-32.

Similarly, IMAXCOL specifies the highest-numbered column in the input line. Before each application of READCH, a check is made to see if ICURCOL > IMAXCOL. If so, the line terminator ENDIN (cf. Section 2.3.1) is invoked. IMAXCOL, like MAXCOL, is constant over the life of a file.

The user is cautioned that IMAXCOL must be exactly equal to the size of lines being read from the designated device, or else file compatibility cannot be guaranteed and data will be garbled.

2.1.2 Vertical Control Variables

2.1.2.1 Top Line (TOP and ITOP)

The fluid variable TOP specifies the top line of an output page, and the fluid variable ITOP specifies the top line of an input page. When page overflow occurs, the function ENDOUTP is executed and causes the line counter CURLINE to be set to TOP and also causes the correct number of lines to be spaced over. Similarly, ENDINP causes the line counter ICURLINE to be set to ITOP and causes the correct number of lines to be ignored.

2.1.2.2 Bottom Line (BOT and IBOT)

BOT designates a warning point for an output page in the same way that RMG designates a warning point for an output line. When the condition CURLINE = BOT occurs immediately after ENDOUT has incremented CURLINE, the BOT overflow function is invoked. In other words, BOT overflow occurs just before line BOT is about to be printed. The BOT overflow function, specified by the user as in Section 1.2.1.6.2, might, for example, cause a trailer to be printed. BOT overflow can be suppressed entirely by setting BOT > PAGE. The behavior of IBOT is entirely analogous.

2.1.2.3 Page Termination (PAGE and IPAGE)

Output page termination occurs when, immediately after CURLINE has been incremented by ENDOUT, the condition CURLINE > PAGE obtains. At this point the page is terminated and the function ENDOUTP is invoked to move to the top of the next page. Similarly, input page termination occurs when ICURLINE > IPAGE with ENDINP invoked to move to the top of the next page.

2.1.3 Specification of Control Variables

The initial values of LMG, RMG, and the RMG overflow function are determined by the HORIZONTAL and OVERFLOW properties in the file description list (cf. Section 1.2.1.6.2). They may be altered for a given file whenever that file is selected for input or output (cf. Section 1.1.3) by setting any of the variables LMG, RMG, RMGO or ILMG, IRMG, IRMGO in section IO.

RMGO and IRMGO are FUNCTIONAL FLUID LOC variables that name the functional constant of no arguments and no value invoked at right margin overflow for output and input, respectively. The other variables are all INTEGER FLUID LOC.

If the condition

$$1 \leq \mathrm{LMG} < \mathrm{RMG}$$

is not satisfied, HORIZONTAL properties have no effect and the default values are used.

Similarly, the initial values of TOP, BOT, and the BOT overflow function are determined by the VERTICAL and OVERFLOW properties in the file description list.

Again, as with HORIZONTAL properties, VERTICAL properties may be altered for a given file whenever it is selected by setting any of the variables TOP, BOT, BOTO, or ITOP, IBOT, IBOTO in section IO. BOTO and IBOTO are FUNCTIONAL FLUID LOC variables that name the functional constant of no arguments and no value invoked at bottom-of-the-page overflow for output and input, respectively. The other variables are all INTEGER FLUID LOC.

The condition

$$1 \leq \mathrm{TOP} < \mathrm{BOT}$$

must always be guaranteed.

2.1.4 Effect of Simultaneous Input and Output on Page Control Variables

If input and output are done simultaneously on the same file, then all of the input control variables coincide with the corresponding output control variables. For example, the variables CURLINE and ICURLINE will both take their values from the same cell. The result is that any change to an input variable will cause a corresponding change in an output variable, and conversely. It also follows that the input page format and output page format will be the same unless they are changed whenever there is a shift from input to output, or conversely.

2.2 READING AND PRINTING FUNCTIONS

The reading and printing functions always operate on the currently selected input or output file. They are thus file-independent, and may be used without knowledge of which file is currently selected, providing the file has the proper FORM. The primitive functions for BCD or ASCII forms are READCH and PRINCH; the other functions are programmed in terms of these. The reading and printing functions do not themselves specify any of the format parameters, but they do make use of the format parameters that already exist.

Certain of these functions are sensitive to symmetric printing; that is, they will print LISP 2 data types (strings and identifiers) with unusual spellings, so that they may be read back by LISP 2 from the output file as the same internal data types from which they were printed. A full discussion of symmetric printing is provided in Section 2.2.9.

As a mnemonic aid, all print primitives that drop the "t" in their names, e.g., PRIN, PRINCH, etc., do not evaluate ENDOUT (cf. Section 2.3) and do not terminate the current line. Those that use the full spelling of "print" in their names do terminate the current line by evaluating a final ENDOUT. In general, the read primitives do not evaluate ENDIN (cf. Section 2.3), and thus do not terminate the current line.

2.2.1 <u>READCH</u>

READCH is a function of no arguments. It reads the current character of the line and returns that character as a one-character identifier; it also increments ICURCOL by one. All format and overflow controls are in effect. If line or record overflow is encountered, the next line or record will be positioned by ENDIN or ENDINR, and the value of READCH will be the ASCII null character (OQ). Thus, READCH can be used to test for line or record boundaries. Since these conditions return OQ, the INTEGER FLUID LOC, variables ISTATUS in section IO, as discussed below, can be used to remove the ambiguity.

READCH will also return OQ if it is impossible to read the next character because a physical (tape) or logical (disc) end-of-file or end-of-medium was encountered. Again ISTATUS can be used to clarify the nature of the read failure. When an end-of-medium condition obtains, READCH locks out further reading. (See Figure 2 for unlocking by means of IKEY.)

2.2.2 <u>PRINCH</u>

PRINCH is a function of one argument, a one-character identifier that is also returned as its value. The character is entered in the line at the current column, and CURCOL is incremented by one. All format and overflow controls are in effect. If line or record overflow is encountered, ENDOUT or ENDOUTR will be evaluated. For symmetry with READCH, PRINCH will accept the null character (OQ) as its argument. The null character is treated by PRINCH as a line termination, i.e.,

PRINCH(null character) ≡ ENDOUT()

PRINCH does not call ENDOUT unless line overflow occurs, or its argument is the null character (0Q).

2.2.3 PRINATOM

PRINATOM is a function of one argument, which is a LISP 2 atom that is also returned as its value. The print name of the atom is entered in the line, starting at the current column. All format and overflow controls are in effect with ENDOUT automatically called if line overflow is encountered. PRINATOM does not normally call ENDOUT, and after evaluation, CURCOL marks the column of the line following the last character of the atom's print name.

Many atoms, such as numbers and Booleans, have no print names. For these cases special primitives, such as TOSTRG and PRINSTRING, will be called to print these atoms. (TOSTRG is a conversion function that converts any atom to a string.)

PRINATOM will apply special primitives for atoms with unusual spellings as described in Section 2.2.9.

PRINATOM will split atoms across line boundaries if they cannot fit on a line. For cases where this is intolerable, the function FITATOM should be used. FITATOM is entirely analogous to PRINATOM but attempts to fit the atom on one line, calling ENDOUT if necessary, prior to entering any characters. FITATOM will probably fail for symmetric printing since the number of characters to be printed cannot be computed prior to printing when a pre-print logic is used.

2.2.4 PRINSTRING

PRINSTRING is a function of one argument, a LISP 2 string, that is also returned as its value. The string is taken literally as its print name and entered in the line starting at the current column, as if each character in the string were printed with PRINCH. All format and overflow controls are in effect with ENDOUT automatically called if line overflow is encountered. When line overflow does occur, the balance of the printed string is printed, beginning in Column 1 of the next line. PRINSTRING does not normally call ENDOUT, and after evaluation CURCOL marks the column of the line following the last character of the string.

PRINSTRING can be used to print a string with unusual spelling so that it has READ symmetry (cf. Section 2.2.9).

2.2.5 PRIN and PRINT

PRIN is a function of one argument, an S-expression, that is also the value of PRIN. Starting at the current column, PRIN enters left and right parentheses, dots (set off with blanks), and print names for all atoms in the S-expression, in list notation format. Atom print names are entered in the line by primitive PRINATOM. PRIN does <u>not</u> provide symmetric printing. PRINT is PRIN plus a final ENDOUT.

All format overflow controls are in effect with ENDOUT automatically called if line overflow is encountered. PRIN does not terminate with a final ENDOUT; therefore, after evaluation, CURCOL marks the column of the line following the last character of the S-expression.

2.2.6 READ

READ is a function of no arguments. Its value is the next S-expression in the file beginning at the current column. If line overflow occurs, READ automatically calls ENDIN. READ does not normally terminate with ENDIN, and therefore the current column is set to the next column following the last character of the S-expression read, when a self-delimiting datum such as a list or string is read. All other format and overflow controls are in effect.

READ operates by combining tokens into atoms and list structures as directed by the structure of the S-expression being read. READ calls upon a finite state machine to supply these tokens. The finite state machine uses READCH to read characters in the file, which it converts into LISP 2 tokens. Thus, READ does not directly concern itself with the processes of searching and maintaining the list of existing identifiers, composing numbers, making strings and the like. These are more efficiently performed by the finite state machine.

2.2.7 SYMPRIN and SYMPRINT

SYMPRIN is a symmetric PRIN (cf. Section 2.2.9) whose argument and value are a LISP 2 S-expression. SYMPRIN is symmetric because it prints the S-expression in such a way that the printout, when read back in by LISP, will yield, internally, the identical S-expression. In all other ways, SYMPRIN is like PRIN. SYMPRINT is SYMPRIN plus a final ENDOUT.

2.2.8 READWORD and PRINTWORD

These two primitives are used with binary files exclusively. ENDIN and ENDOUT for READWORD and PRINTWORD, respectively, are always invoked since a word and a line are equivalent for binary files.

READWORD is a function of no arguments, whose value for Q-32 LISP 2 is a 16-digit octal number contained in the current word of the octal array for the record. READWORD advances control to the next word, after evaluation.

PRINTWORD is a function of one argument, an octal number, that is returned as the value of the function. PRINTWORD enters the octal number into the current word of the octal array for the record and advances control to the next word.

2.2.9 Symmetric Printing

The identifier "ABC" will always print as "ABC" and will always be read as the identifier "ABC" since there is nothing unusual about the spelling of the identifier. On the other hand, the spelling "AB)C" can never be read as an identifier because of the unusual spelling (the contained parenthesis is not a period, letter, or number). If one wishes this spelling for the identifier, he must spell it as "%#AB)C#". But on output, normal printing will yield "AB)C" and again one can not read the spelling back in as an identifier. Similarly, if one wishes to read the characters "AB)C" as a string he must spell it as "#AB)C#". Normal printing of this string yields "AB)C", and here also one can not read the spelling back in as a string, its original internal form. For these reasons, one must have a means for telling LISP 2 to print unusually spelled identifiers and strings with read symmetry.

PRMODE (print mode) is a fluid Boolean system variable in section SYS used to tell the print primitives PRINSTRING and PRINATOM how to print unusually spelled identifiers and strings. If PRMODE = TRUE, symmetric printing is requested. If PRMODE = FALSE, normal printing is desired. PRMODE is set to FALSE at the top level.

When PRIN (PRINT) is evaluated, PRIN locally binds PRMODE to FALSE before calling PRINSTRING and PRINATOM to print elements of an S-expression. In this way, regardless of the higher-level binding of PRMODE, normal printing is guaranteed. Conversely, SYMPRIN (SYMPRINT) binds PRMODE locally to TRUE to guarantee symmetric printing. Thus, the user may use PRIN (PRINT) or SYMPRIN (SYMPRINT) to print S-expressions normally or symmetrically without concern with the state of PRMODE. However, if he uses PRINSTRING or PRINATOM and is concerned with which print representation he gets, he should bind PRMODE appropriately prior to evaluating PRINSTRING or PRINATOM.

2.3 TERMINATOR FUNCTIONS

There are six terminator functions, as follows:

	input	output
line	ENDIN	ENDOUT
page	ENDINP	ENDOUTP
record	ENDINR	ENDOUTR

All of these are functions of no arguments, and no value. They are evaluated for their effect on the page control variables. For files with BINARY form only ENDINR and ENDOUTR have meaning.

2.3.1 Line Terminators

The line terminator ENDOUT terminates printing of the current line of the output file and advances the control variables to the next line of the file. ENDOUT causes the following actions:

(1) Fill the current line with blanks up to MAXCOL.
(2) If the record is full, call ENDOUTR.
(3) Increment CURLINE by 1.
(4) Set CURCOL to LMG.
(5) If CURLINE = BOT, call BOTO.
(6) If CURLINE > PAGE, call ENDOUTP.

The line terminator ENDIN is entirely analogous; characters on the current line that have not yet been read are ignored.

2.3.2 Page Terminators

The page terminator ENDOUTP terminates printing of the current page of the output file and advances the control variables to the top line of the next page. ENDOUTP calculates the number of blank lines needed and calls ENDOUT an appropriate number of times. CURLINE is then set to TOP. The behavior of ENDINP is analogous.

2.3.3 Record Terminators

The functions ENDINR and ENDOUTR are seldom called explicitly by the user, but rather are called by other input-output functions such as the line terminators. When they are evaluated they terminate the file record currently in core. For ENDINR, that means loading another record from the external medium; for ENDOUTR, it means dumping the current core record onto the external medium. To achieve these operations it is necessary for the record terminators to invoke string post-processes that:

(1) Read or write a raw external core image from or to the external medium.

(2) Delete or insert requisite format information.

(3) Translate raw character codes from 12 to 8 bits, 8 to 12 bits, 6 to 8 bits, 8 to 6 bits, from tape, teletype, and disc standards.

The required string post-processing is determined from the file description.

ENDINR and ENDOUTR perform essentially the same actions (though in a different order) and are explained here in terms of output variables:

(1) Clear record-string (to blanks for symbolic files, to zero for binary files).

(2) Load or dump external core image.

(3) Perform string post-processing.

(4) Set CURCOL = LMG (for symbolic files only).

Note that the record terminators do not invoke the line terminators so that any partially filled line may be lost after a record terminator is evaluated. Also note that the line terminators invoke the record terminators only when record overflow is encountered. When reading blocked records, it is sometimes desirable to explicitly evaluate ENDINR before a record is fully read, thus initializing the file controls to the first line of the next record. When printing blocked records, a final ENDOUTR should be evaluated before quitting for the day, or before SHUT is evaluated; otherwise, any data in a partially filled record will be lost.

2.4 POSITION

POSITION is a function of two arguments that positions the current record of the file. It also allows various termination marks (e.g., end-of-file, end-of-tape) to be written in the file. It has the form:

```
POSITION(name, action)
```

where name is the name of an available file, and action is a reserved input-output variable that describes the position action desired, e.g., rewind, skip file, etc. The variable evaluates to a positive decimal integer, the action code for a file operation. The value of POSITION depends on the action desired.

For many files, particularly TTY, CRT, and CORE units, POSITION acts as a NOP with a value of NIL. NIL is also returned for illegal action codes. For POSITION, a record corresponds to a record on tape, and a sector on disc. Furthermore, many physical files, separated by an end-of-file mark, may exist on tape; however, only one file is permitted on disc. Therefore, it is impossible to POSITION beyond a disc end-of-file, but you can POSITION past a tape end-of-file at your own risk.

The legal action codes, values, and their meanings are given in Figure 2.

2.5 FORMAT INTERROGATION

All input-output control variables are available for examination or change as FLUID LOC variables of various types in section IO. Therefore, by "tailing" to section IO one has easy access to these variables. For example, one could define a new function TABO that advances the current column of the selected output file to some desired legal column and returns the original column as output.

```
INTEGER FUNCTION TABO (COL)      INTEGER COL

 BLOCK (X←CURCOL$IO) INTEGER X:

   IF      COL>MAXCOL$IO OR COL<1 THEN GO EXIT ELSE

           CURCOL$IO←COL;

EXIT: RETURN X;

END;
```

Not all IO variables are modifiable and caution should be used in setting or binding them. Figure 3 is a complete statement of these variables and their use.

Action Variable	Value	Action	Value
SKIPR.	1	Skip to line 1 of next record	. EOF, if next record is an end-of-file
SKIPF.	2	Skip to line 1 of next tape file--or to the end-of-file mark of this disc file	. n, the number of non-EOF records skipped . EOT, if next record is an end-of-tape . If EOF encountered, tape positioned after EOF mark, disc positioned at EOF mark (can be overwritten)
WEOF.	3	Write an end-of-file mark at current line of file	
WEOT.	4	Write an end-of-file mark at current line of tape file; action 3 for disc files	
REWIND.	5	Backup to line 1 of file (rewind)	
BACKR.	6	Backup to line 1 of prior record	file name
BACKF.	7	Backup to last line (just before end-of-line) of prior tape file; action 5 for disc files	
KEY.	8	Unlocks READCH. for further reading. READCH locks out further reading if an end-of-medium is ever encountered	

Figure 2. Action Codes for POSITION

Name	Type	Range of Value	Use
LMG(ILMG)	INTEGER	1-MAXCOL	Left-most column of a line--margin control
RMG(IRMG)	INTEGER	1- ∞	Right-most column of a line--margin control
TOP(ITOP)	INTEGER	1-PAGE	Top-most line of a page--margin control
BOT(IBOT)	INTEGER	1- ∞	Bottom-most line of a page--margin control
PAGE(IPAGE)	INTEGER	1- ∞	Maximum line of a page
CURCOL(ICURCOL)	INTEGER	1-MAXCOL	Current column of a line
CURLINE(ICURLINE)	INTEGER	1-PAGE	Current line of a page
SUMLINE(ISUMLINE)	INTEGER	1-RECORD	Current line of a record
STATUS(ISTATUS)	INTEGER	1- ∞	Status code for last output (input) record transfer according to the following: 1 = end of line 2 = end of record 3 = end of file 4 = end of medium greater than 4 = error condition
SECTOR(ISECTOR)	INTEGER	0-(8n-1)	Current disc sector being accessed. There are 8 sectors/track, where n = track number.
SIZE(ISIZE)	INTEGER	1- ∞	Generally means number of words read into core in last ENDINR for tape or disc files.
COUNT(ICOUNT)	INTEGER	100-100n	Next card number, n, times 100 for Q-32 disc output files. Required for non-LISP service routine compatibility.
RMGO(IRMGO)	FUNCTIONAL	No value No argument Function name	Right margin overflow function
BOTO(IBOTO)	FUNCTIONAL	No value No argument Function name	Bottom line overflow function

Figure 3. Section IO Control Variables

Name	Type	Range of Value	Use
KEY(IKEY)	FUNCTIONAL	No value No argument Function name	Function executed by READCH if file locked
(The following variables are used internally by LISP 2 input-output and should never be changed by a user after a file has been opened.)			
MAXCOL(IMAXCOL)	INTEGER	Device-dependent	Maximum column of a line
RECORD(IRECORD)	INTEGER	Device-dependent	Maximum line of a record
WPL(IWPL)	INTEGER	Device-dependent	Number of machine words/line
NAME(INAME)	INTEGER	Variable	An integer that in octal represents the first 6 characters of the BCD file name.
MAXSEC(IMAXSEC)	INTEGER	8n-1	The maximum possible disc sector allocated for file; n = number of tracks allocated.
MOVE(IMOVE)	FUNCTIONAL	No value No argument Function name	String post-processing function
XXFUNC	FUNCTIONAL	Symbol value No argument Function name	Function used by the finite state machine to get next character when called by READ
XXSAVE	SYMBOL	Character atom or NIL	Delimiter saved by the finite state machine during token parsing for backup
CURFN(ICURFN)	SYMBOL	File name	Currently selected file name
CPW	INTEGER	6	Machine dependent parameter for ASCII character packing/word
BUFLOC(IBUFLOC)	ARRAY OCTAL FLUID	Variable	Array name for internal string record

Figure 3. Section IO Control Variables (Continued)

Name	Type	Range of Value	Use
BUFIX	ARRAY OCTAL FLUID	Variable	Array name for internal IO buffer
LINELOC(ILINELOC)	OCTAL	Variable	Internal address of the line indicated by SUMLINE
FIXLOC	OCTAL	Variable	Internal address of header word of internal IO buffer

Figure 3. Section IO Control Variables (Continued)