12 June, 1979

Greetings:

I am sending you the attached report either because (1) you have expressed interest in the DICE text editor, or (2) I think you might find it interesting.

The system is written entirely in APL, which means that at present it runs on the APLplus system on the 370/155 at Syracuse University. It depends heavily on the filing system designed by Scientific Timesharing Corp. and I. P. Sharp Asso. As we will be acquiring a new APL system this summer, from I. P. Sharp, DICE will be upgraded to run on it, using system variables and functions.

I would welcome any comments you might have, either on the documentation or the editor proper.

[illegible]

1-9 widow -- bad

2.0.2 "

2.2.1 pad inside of quoted string
line -7 should be an option to [illegible] this -

DICE

A TEXT AND PROGRAM EDITOR

LA16-1231

AUTHOR:	DANA E. CARTWRIGHT 3rd
PUBLISHED:	JUNE 1, 1979
LEVEL:	ADVANCED
COMPUTER:	IBM SYSTEM 370/155
REFERENCES:	SUPERSEDES LA16-1022
PRICE:	$2.00

80 Pages

Contents

Introduction

DICE is an editor which may be used for:

- Document preparation. Initial input of text, editing, and final printout are all supported. This document was prepared and printed using DICE.

- Program source statement preparation. Intended primarily for IBM System 370/155 based batch language processors, DICE supports initial entry of program source, editing, and submission. No provision is made for monitoring the progress of the job or examining its output at the terminal. DICE thus functions as a sophisticated keypunch for this application.

- *APL* file editing. Since DICE uses *APL**plus files to store text, it may prove a convenient tool for working directly with such files.

- User-specific applications. DICE provides the user with the ability to write customized applications in *APL* with DICE supplying text-editing capabilities. Two approaches are possible: user-written functions can be loaded into the DICE workspace and executed in the DICE environment, or the DICE workspace can be invoked from other *APL* workspaces. Details are given in Appendix A.

(Note that no knowledge of *APL* proper is required to use DICE for the first two applications.)

The editor works upon text stored in *APL* files. The text is organized into lines of up to 132 characters each. For purposes of program source preparation, each line may be considered to be a "card image". For document preparation, facilities are provided for producing neatly formatted output from a file which contains text and control information.

<u>GETTING STARTED</u>

To use the editor, sign on to *APL* and load the workspace 2 *DICE*. The editor will automatically begin running. If you have not run the editor before, some questions will be asked about your account number. The answers to these questions will be recorded by DICE for future reference. See Appendix D for an example of using DICE for the first time.

To successfully run the editor under an *APL* account, the file quota of that account must be at least one (1). However, since each file created by the editor is stored as a separate *APL* file, it is desirable to make the quota somewhat larger.

Files are created by typing lines of text on a terminal. Each line is captured by the editor and placed into the <u>workfile</u>. Once the text has been entered it may be (1) sent to the IBM System 370/155 as though it were a card deck, or (2) printed out at the terminal as a document, or (3) saved as a <u>library</u> file with a name identifying it so that it can later be retrieved, or (4) altered and corrected using the facilities of the editor. Each user of the system can create an essentially unlimited number of library files.

INPUT CONVENTIONS

For each line of input typed at the terminal, DICE must decide whether it is text to be placed into the workfile or a command which must be acted upon immediately. The distinction is made by requiring that every DICE command begin with a special, reserved character, which is called the discriminative character ("dister" for short) because it serves to discriminate between normal text and editor commands. The first character typed on the line is inspected -- if it is the user's dister, then the line of input will be interpreted as a command; otherwise, it will be inserted into the workfile as text.

When DICE is first run under an account the period (.) character is used as a dister. Throughout this document all commands will be shown as beginning with a period. Commands are provided for selecting some other character for the dister, however, and the user should select some other character if "period" is unsatisfactory.

More than one command may be typed on a line. The left-most one is executed first, then the next one in sequence, etc. If an error occurs in any command, then the commands to its right are not executed. This feature is called "command stacking".

FILES

A file is a collection of lines of text, numbered from 1 to the number of lines in the file. This numbering is dynamic, meaning that any change in the number of lines in a file also causes simultaneous renumbering of all lines beyond the point of change.

All editing and input is performed against a distinguished file called the workfile. Library files are edited by first being copied into the workfile. This provides a degree of protection against errors since library files are never directly modified. The worst that can happen as a result of an erroneous editing action is that the workfile is destroyed.

The workfile may have associated with it a file name, which may or may not coincide with the name of a library file. Normally when a library file is copied into the workfile for editing, the workfile "name" is set to the name of the library file. This is done merely so that the user can easily remember which library file is being edited. Subsequently the workfile can be copied back into the library file without bothering to tell the editor the proper library file name to use, since the editor simply uses the name associated with the workfile. Beyond that, however, there is no special relationship between the workfile and the library file with the same "name".

WORKFILE INTEGRITY

The editor is designed so that system crashes cause a minimum amount of data to be lost. Specifically, the contents of the workfile at the time of crash are always recovered when the editor is next run, unless system damage to *APL* has occurred. Moreover, several steps have been taken in the design and implementation of DICE to insure that the contents of the workfile are as nearly up-to-date at all times as is consistent with minimization of disk I/O. Every time that the current in-core text buffer is backed up (copied) onto disk, the symbol

```
→→HH.MM←←
```

is printed, where "*HH.MM*" refers to the current time. After a crash the terminal output can be examined to determine the most recent point at which the backup symbol was printed -- all editing done after that has been lost.

APL FILE STRUCTURE

The files generated by the editor consist of vectors of text with carriage returns embedded to delimit lines. The text may be broken up into many components, but this is done in such a way that catenation of successive components produces correct text vectors (that is, each component terminates with a carriage return, and the components are properly ordered with respect to the text). Files not created by the editor but satisfying the above conditions may be loaded by the editor. Additionally, the file components may consist of matrices of text, and the editor will interpret each matrix row as a line of text. However, the matrix structure of the file will be lost once it is subjected to the *SAVE* command. All commands which reference files will work upon either vectors or matrices.

ATTN KEY

The editor may be interrupted at any time by pressing the ATTN (BREAK on some terminals) key. In general this results in some one of the editor functions suspending (being interrupted), in the *APL* sense of that word. This results in a line of output on the terminal of the form

```
name[n]
```

where "name" is the name of an *APL* function and "n" is a decimal number. Type

```
→n
```

to restart the editor, where "n" is the same number as was printed out in the previous line.

Following such a restart the editor will terminate execution of the current command and will print a message to indicate just which command was terminated. In general the command currently in execution does not terminate immediately -- that is, it may continue to run briefly after a restart. Some commands (*SAVE*, for example) will simply continue to completion after being interrupted and restarted.

Command Summary

The table below lists all DICE commands and gives a summary of their usage. This is intended for reference only and might well be skipped by first-time readers.

A	Appends text to the right-hand end of a line.
APL	Executes an *APL* expression using ⍎.
ASEQ	Automatic Sequencing. Assigns an 8-digit sequence number to each line of input to the workfile.
B	Bottom. Makes the last line of the file the current line.
BELL	Changes the state of the keyboard "bell" prompt.
C	Changes text by contextual substitution.
CARDS	Changes estimate of punched card output on a JOB card.
CLASS	Changes CLASS= parameter on a JOB card.
CLEAR	Clears the workfile.
CLOSE	"Closes" the Internal Job Reader, finishes off a job submittal.
COMMANDS	Prints a list of all DICE commands.
CONTINUE	Signs off DICE and *APL* in such a way that at the next sign-on to *APL*, DICE will automatically be loaded and the current session continued.
COPY	Copies text into the workfile, from a library file or elsewhere in the workfile.
COST	Shows cost of current DICE session.
D	Down. Moves current line pointer towards the end of the workfile.
DISTER	Changes the Discriminative Character.
DO	Executes DICE commands from a file.
E	Erases lines of the workfile (see also *ZAP*).
FILES	Lists names of DICE files.
FORM	Prints a "form" on the page, for assistance in typing input to align with a form.
FORMAT	Used in conjunction with the "*PF*" command, prints a formatted

	document.
FORMS	Lists all available forms (see the "*FORM*" command).
GANG	Sets/clears "gang" punching on type *ASM* files.
HELP	Provides on-line command documentation.
HOLD	Pauses DICE while forms are adjusted in the terminal.
I	Inserts text into the workfile.
J	Joins two lines of text into one.
L	Locates a line by string context.
*LA*16	Prepares this document.
LINES	Changes the estimate of printed output lines on a JOB card.
LIST	Lists a file at the terminal.
LOAD	Loads a library file into the workfile.
MOVE	Moves text about in the workfile.
NAME	Changes the programmer name on a JOB card.
OFF	Signs off DICE, returns to *APL*.
P	Prints a line of the workfile at the terminal.
PA	Asks for a Public Announcement of DICE to be reprinted.
PLOT	Changes the plotter time estimate of a JOB card.
PRINT	Prints a file on the high-speed line printer.
PRINTN	Prints a file on the high-speed line printer, with line numbers.
PSWD	Changes the default JOB card password.
PUNCH	Punches a file onto cards.
PURGE	Cancels a Job which has been partially submitted to the Internal Job Reader.
R	Replaces text in the workfile.
REGION	Changes the REGION= parameter on a JOB card.
RUN	Submits a file as a Job to the IBM System 370/155.

S	Splits a line into two lines.
SAVE	Saves the workfile as a library file.
SE	Super Edit. Edits a line from the workfile in a fashion similar to the *APL* function editor.
SEQ	Writes sequence numbers (a la a card deck) on lines of the workfile.
SHARE	Permits other *APL* users to access a library file.
T	Top. Makes the first line of the workfile the current line.
TABS	Specifies a uniform tab setting for the terminal.
TERMINAL	Tells DICE the type of terminal being used, for document printing only.
TEXT	Moves text from the *APL* environment into the workfile.
TIME	Changes the equivalent time estimate on a JOB card.
TYPE	Changes the workfile type.
TYPES	Prints a list of all valid workfile types.
U	Up. Makes the previous line the current line.
V	Verify. Prints a message to the user and proceeds only when so directed. Used in *DO* files.
WAIT	See the *HOLD* command.
WFID	Changes the workfile id.
ZAP	Deletes a library file.
?	Asks about current line number and number of lines in the workfile.
⎕	Suspends DICE, returns to *APL*, in a way which permits DICE to be immediately restarted.
=	Same as the *DISTER* command.
\	Invokes a user-written *APL* function as a command.
*	Writes a message to the terminal. Used in *DO* files.
)*LOAD*	Loads a workspace, thus leaving DICE abruptly.
)*OFF*	Signs off DICE and *APL*.

All of the DICE commands are presented below in detail. They are grouped into these categories:

Line Pointer Control
Editing and Printing
File Management
Program Preparation, JCL, Batch access
Typed Files
Terminal Control
Session Control
APL-related
Miscellaneous

The commands are described informally, using many examples of specific command usage. A complete specification of command syntax is given in Appendix B (intended primarily for advanced users of DICE).

Line Pointer Control Commands

One line in the workfile is always distinguished and is termed the current line. Most editing commands work implicitly upon this line. Imagine that a line pointer exists which "points to" the line of the workfile which is to be the current line. The following commands are used to set the line pointer, that is, select the "current line".

.T

Makes the first line of the workfile (top) the current line.

.B

Makes the last line of the workfile (bottom) the current line. Thus, if there are 15 lines of text in the workfile this command sets the current line pointer to 15.

.U

Moves the current line pointer back by one line (up, towards the first line of the file). It may be followed by a number, in which case the line pointer is moved backwards by that number of lines. The current line pointer can be moved to line 0 by this command.

.D

Advances the current line pointer by one line (down, towards the last line of the file). It may be followed by a number, in which case the line pointer is advanced by that number of lines. If a request is made to advance the pointer beyond the end of the file the response to the command is "*NO*".

.*?*

Prints the current line number and the total number of lines in the file. For example,

```
    .?
    5 OF 8
```

.*L*

This command is used to advance the current line pointer until a specified string of text is located. Thus, for example,

```
    .L/HOHOHO/
```

causes the current line pointer to be advanced until a line containing the character string "*HOHOHO*" is found. The "*P*" modifier should be used if it is desired to print out the line so found. For example,

```
    .L/HOHOHO/P
```

If the string is omitted then the string from the previous "*L*" command will be used. For example, if the command indicated above had previously been used in this editing session, then the command

```
    .LP
```

would search for the string "*HOHOHO*" (and would print out the line containing it).

To find all lines in a file which contain a given string of text, the command may be used with the "*M*" (multiple) modifier. This modifier causes the command to continue on after finding a line containing the specified string. For example,

```
    .L/HOHOHO/MPN
```

will find the next line of the workfile which contains the string "*HOHOHO*", and will print it out along with its line number (because of the "*P*" and "*N*" modifiers), and then will continue on through the file, repeating these actions, until the end of the file is reached.

The "*M*" modifier may be followed by a number, in which case the command quits after that many lines containing the string have been found. For example,

```
    .L/HOHOHO/M3PN
```

will find the next 3 lines which contain the string "*HOHOHO*".

These commands are used to display and alter all or parts of the workfile. Several of the display commands may be used with library files as well.

.*P*

This command is used to print out selected lines of the workfile. When used with no modifiers (that is, the command consists of just the letter "*P*"), the current line is printed out.

A series of contiguous lines, beginning with the current line, is printed out by specifying a line count with the command. For example,

```
.P10
```

prints out 10 lines, beginning with the current line.

<u>The current line pointer is not changed by the command.</u>

The entire workfile may be printed by selecting the first line as the current line and then using the command

```
.P*
```

The "*" indicates that the file is to be printed from the current line through the last line of the file.

The column modifier may be used to restrict the output to selected columns. For example,

```
.P5C10-15
```

lists columns 10 through 15 (inclusive) of 5 lines, beginning with the current line.

See Section 3 for details on using the *P* command for formatted output.

.*I*

This command inserts text into the workfile. In its simplest form no operand is specified (that is, the command consists of just the letter "*I*"). The editor responds with the message "*INSERT, EXTRA CR TO EXIT*" and the keyboard unlocks for input. Each line typed is inserted into the file following the current line, and the inserted line becomes the new current line. When an empty line (just RETURN) is typed, the insert command ends.

Alternatively, the text of the line to be added may be included with the command. For example

```
        .I/ADD THIS LINE/
```

causes a new line of text to be inserted into the workfile, consisting of the text "*ADD THIS LINE*". The inserted line becomes the new current line.

To insert a number of identical lines the above form of the command may be used with a repetition count. For example,

```
        .I/ADD THIS LINE/L10
```

Adds 10 lines (all identical) to the file, following the current line. The last line added becomes the new current line.

In addition to adding full lines, the command may be used to add text to an existing line. This is done by specifying a column modifier designating where in the line the text is to be added. For example, suppose the current line of the workfile is

```
        Now is the time
```

and the command

```
        .I/NOT /C7
```

is executed on the line. The line will then appear as

```
        Now is not the time
```

that is, the text was added after column 7. Note that material may be placed at the front of a line by specifying the column modifier "*C0*"; that is, add the text "after column 0". See the "*A*" command for details on how to add text to the end of a line.

The "*P*" modifier may be used with the command, in which case each inserted (or modified) line is printed out.

.R

Replaces lines of the workfile. For example,

```
        .R/HOW NOW/
```

deletes the current line and replaces it with "*HOW NOW*".

When used with a column modifier, this command may also be used to replace selected parts of lines. See the "*I*" command for details on using column modifiers.

.*E*

This command is used to erase lines of the workfile. Only the current line is erased unless some other number of lines is specified. For example,

```
.E5
```

requests that 5 lines of the workfile be deleted, beginning with the current line.

Columns (rather than lines) of text may be deleted by using a column modifier. For example,

```
.EC5
```

erases "column 5" (the 5th character) of the current line.

```
.EC5+3
```

erases 3 columns, starting in column 5 (erases the 5th through the 7th characters).

```
.EC5-8
```

erases the 5th through the 8th column. In this latter form, either the starting or the ending column specifier may be elided; the command then operates from the first (or through the last) column. For example,

```
.EC17-
```

erases column 17 through the end of the line.

A conditional form of the command is available and may be invoked via the "*V*" (verify) modifier. Thus

```
.EV
```

is the same as just the "*E*" command with the addition that text to be erased is first displayed and the keyboard unlocks for input. Replying "*Y*" or "*YES*" continues with the erasure, otherwise the workfile is unchanged.

When four or more lines are erased the command functions as though a "*V*" modifier had been used.

.C

This command is used to edit (Change) text in the workfile. Two strings of text are given, called the "new" and "old" string. The first occurrence of the old string in the current line is replaced with the new string. The strings need not be of the same length. For example, if the current line is the text

```
HOW NOW BROWN COW
```

then the command

```
.C/BROWN/BLUE/
```

will change "*BROWN*" to "*BLUE*", leaving the line as

```
HOW NOW BLUE COW
```

If the new string is empty, then the old string is merely deleted (i. e., replaced by nothing). If the old string is empty, then the most recent string used with the "*L*" command is used. For example,

```
.L/ABC/
.C//DEF/
```

is equivalent to

```
.L/ABC/
.C/ABC/DEF/
```

As outlined above the command works only upon the current line. If the "*L*" modifier is used, however, successive lines of the workfile are examined until one is found which contains the "old" string. The substitution is then made as outlined above and the command stops. The modified line becomes the current line. For example,

```
.C/ABC/DEF/L
```

works in this way.

If <u>both</u> strings are omitted, then the strings from the previous "*C*" (not "*L*") are used instead. For example, the two commands

```
.C/ABC/DEF/
.C
```

are equivalent to

```
.C/ABC/DEF/
.C/ABC/DEF/
```

The new form of the modified line may be displayed by using the "*P*" modifier with the command. For example,

```
    .C/ABC/DEF/P
```

will display the line with "*DEF*" substituted for "*ABC*".

Normally only the first occurrence of the old string is modified. The "*M*" (multiple) modifier may be used to select some other option. For example,

```
    .C/ABC/DEF/M
```

will cause substitution for all occurrences of "*ABC*" in the current line. If a number is specified after the "*M*", then only that many occurrences are changed, counting from the left-hand end of the line.

If it is desired to inspect the line before changing it, the "*V*" (verify) modifier should be used. This is often useful when used in conjunction with the "*L*" modifier. For example,

```
    .C/ABC/DEF/PLV
```

will

1. Inspect the current line (and subsequent lines if necessary, because of the "*L*") for the string "*ABC*".

2. Print the line containing "*ABC*" in a condensed format (the first and last few characters).

3. Request input. Reply "*Y*" if the substitution is actually to be made. Otherwise, just press RETURN. Because of the "*P*" modifier, if the substitution is actually made, the altered form of the line will be displayed (in full).

.SE

This command is a variation on *APL* "Super Edit" (the *APL* function editor). The current line is printed out and keyboard input is requested. Typing a slash (/) under a character deletes that character. Any other character typed on the line is merged into the displayed line, except for the blank and quote. Here are some examples:

```
DISPLAYED:     This is a line.
INPUT:         ////there     / here
RESULT:        there is a line here

DISPLAYED:     A=A-1*4
INPUT:         /C/C   6
RESULT:        C=C-1*46
```

```
DISPLAYED:     This line will be shortened.
INPUT:                   ///////was
RESULT:        This line was shortened.
```

Neither the blank nor the slash character can be directly added to a line with this command. To overcome this restriction the edit line can contain quoted strings, where "quoted" means strings enclosed in single quote (') characters, and not the special type of string used in certain DICE commands. Quote marks are included in a quoted string by doubling them. Some additional examples:

```
DISPLAYED:     5000  FORMAT(3I5)
INPUT:                      /'2I5,'' THE VALUE OF I IS ''',
RESULT:        5000  FORMAT(2I5,' THE VALUE OF I IS ',I5)

DISPLAYED:     K=(I*J)+(J+1)*(I+1)+(J-1)-(I-1)*J
INPUT:           (         /') * '     /+    /'/'
RESULT:        K=((I*J)+(J+1)) * (I+1)+(J-1)+(I-1)/J
```

.A

This command is used to append text to lines in the workfile. The current line will be displayed, and the carriage will remain at the right-hand end of the text. Additional text may then be typed which will be appended to the line, or the line may be edited by using BACKSPACE and ATTN.

Alternatively, the text to be appended may be included directly with the command. For example,

```
      .A/FOO/
```

will append "*FOO*" to the end of the current line.

The command can be applied to multiple lines of the workfile through use of the "*L*" modifier. Thus,

```
      .AL7
```

has the effect of an "*A*" command applied to seven lines of the workfile in succession, beginning with the current line.

.S

Splits the current line into two lines. A string of text is specified and the first occurrence of that string in the current line becomes the beginning of the second line. For example, if the current line is

```
      this line will be made into two lines
```

then the command

```
      .S/made/
```

would change the current line into two lines, namely

```
      this line will be
      made into two lines
```

with the first line becoming the current line.

If the "*P*" modifier is added to the command the two new lines will be displayed, thus

```
      .S/made/P
      this line will be
      made into two lines
```

If the "*V*" modifier is added to the command, it functions as though the "*P*" modifier had been used, plus one has the option of not actually splitting the line. For example,

```
      .S/made/V
      this line will be
      made into two lines
      OK?
```

After "*OK?* " the keyboard will unlock for input. Reply "*Y*" or "*YES*" if the split is actually to be done. Any other reply causes the workfile to remain unchanged.

.J

This command is used to "join" the current line with the previous line. For example, if

```
      Now is the time for all
      good men to come to the aid
```

are two consecutive lines in the workfile, and the second is the current line, then the "*J*" command will join the two lines into

```
    Now is the time for allgood men to come to the aid
```

which will be the new current line.

Since it is often the case that joining the lines directly causes text to run together, the command has the option of inserting additional text between the two lines as they are joined. For example,

```
    .J/ jolly /
```

would join the two lines indicated above to produce

```
    Now is the time for all jolly good men to come to the aid
```

.SEQ

This command writes 8-digit "sequence numbers" in column 73-80 of selected lines of the workfile. For example,

```
    .SEQ 1000
```

writes "00001000" into columns 73-80 of the current line.

```
    .SEQ 1000 FOR 10
```

writes "00001000" into the current line, "00001001" into the next line, etc., for a total of 10 lines. If an increment other than 1 is desired, the form of the command is (for example)

```
    .SEQ 1000 FOR 10 BY 5
```

The "*FOR*" and "*BY*" phrases may appear in either order.

.ASEQ

This command is used to place a "sequence number" on each new line added to the workfile. A "sequence number" is an 8-digit number in character positions 73 through 80 of the line. The operand of the command is a number which will be placed as a sequence number on the next line inserted into the workfile. Each subsequent line which is entered will have a sequence number which is 1 greater than the previous one, unless a different increment is specified via the '*BY*' construct. For example,

```
    .ASEQ 340
```

will place "00000340" on the next line added to the workfile, "00000341" on the second line added, etc.

```
      .ASEQ 340 BY 5
```

will also start with "00000340" but will continue with "00000345", "00000350", etc.

```
      .ASEQ OFF
```

terminates automatic sequencing of input lines.

```
      .ASEQ
```

displays the current automatic sequence number without changing it.

When the command is used to change the state of automatic sequencing, the previous state will be printed. For example,

```
      .ASEQ 500
      WAS OFF

      .ASEQ 600
      WAS 500

      .ASEQ OFF
      WAS 600
```

When automatic sequencing is in effect the sequence number will appear as a prompt whenever keyboard input of a line is requested. For example, after

```
      .ASEQ 1000
```

the next input line will be prefaced (by DICE) with

```
      00001000>
```

Normal DICE commands may still be entered (so long as they begin with the correct dister character). The sequence number displayed is moved into the correct position when the line is installed into the workfile.

See the "*SEQ*" command for information on how to apply sequence numbers to existing lines of the workfile.

.COPY

This command is used to copy text into the workfile, either from a library file or from elsewhere in the workfile itself. The text is always inserted into the workfile just after the current line, and the current line pointer is then advanced to point at the last copied line. Thus

```
.COPY 40-50
```

copies lines 40 through 50 (inclusive) of the workfile. Note that lines 40-50 will still exist unchanged in the workfile. See the "*MOVE*" command for details on how to move lines of text about in the workfile (as opposed to copying them).

```
.COPY JUNK 50-55
```

will copy lines 50 through 55 of library file "*JUNK*" into the workfile.

```
.COPY 55,JUNK -40
```

will copy line 55 of the workfile, then copy the first 40 lines of library file "*JUNK*" (since the starting line number of the copy from the library file is not specified, it begins with line 1).

.MOVE

This command is used to move lines of text about in the workfile. A single line or a range of lines is moved. The text is always moved so that it immediately follows the current line. For example, consider a workfile consisting of these lines:

```
Line 1
Line 2
Line 3
Line 4
Line 5
Line 6
Line 7
```

If line 2, consisting of the text "Line 2", is the current line, and the command

```
.MOVE 4-5
```

is entered, the workfile will then consist of these lines:

```
Line 1
Line 2
Line 4
Line 5
Line 3
Line 6
Line 7
```

If a single line is to be moved, then just its line number may be specified. If the first number of the range is omitted, then 1 is assumed. If the second number is omitted, then "through the end of the file" is assumed. Examples of such commands:

```
.MOVE 4

.MOVE -4

.MOVE 4-
```

.LIST

This command prints a file out at the terminal. If no file is specified, then the workfile is listed. For example,

```
.LIST HOO
```

prints out all of the DICE file "*HOO*" at the terminal. Specific lines of a file may be listed, and more than one file may be included in a single command. Thus

```
.LIST HOO 40-45,FOO,GOO 70-
```

causes lines 40 through 45 (inclusive) of file "*HOO*" to list, followed with no break by all of file "*FOO*", followed by lines 70 through the end of file "*GOO*".

.PRINT

This command sends a file to the high-speed line printer for printing. Lines of up to 132 characters can be accommodated. The output from the printer will be much the same as if the file had been displayed on the terminal using the "*LIST*" command. If line numbers should be included on the output use the "*PRINTN*" form of the command. Some examples:

```
.PRINT
```

prints the contents of the workfile on the line printer.

```
.PRINT 50-75
```

prints lines 50 through 75 of the workfile on the line printer.

```
.PRINTN JUNK
```

prints file "*JUNK*" on the line printer, with line numbers.

The actual printing of the file is done by a batch job. Therefore, this command may only be used if there is a valid batch account available to the user. The accounting parameters and naming conventions used with the batch job are:

Job name - composed from the workfile id (see the "*WFID*" command).

Account Number - the same as the *APL* account number under which DICE is being run.

Password (for the batch account) - as specified by the "*PSWD*" command, which see.

Programmer name - the name which was specified in response to a question asked when the DICE editor was first run under your account (see Appendix D).

Copies - just 1 copy will be printed. This cannot be changed.

.*PUNCH*

This command sends a file to the high-speed card punch to be punched onto 80-column cards. Each line of the file is punched onto a single card. Lines longer than 80 characters are truncated, while shorter ones are padded with blanks.

The file to be punched is specified as in the "*PRINT*" command.

The actual punching is done by a batch job. See the "*PRINT*" command for details on how this batch job will be run.

.*FORM*

This command prints a "form" at the terminal, primarily to assist in typing or editing files in which certain column alignment is desired. A specification of the form desired must follow the command. Thus

```
    .FORM 80
```

prints "123456789|" over and over, for a total of 80 characters across the page. This may prove useful in typing images of card decks or in JCL preparation and editing.

A list of all currently valid form names is given by the "*FORMS*" command. For example,

```
        .FORMS
        80
        PR
```

.LOAD

This command discards the current contents of the workfile and fills it with a copy of specified library files. For example,

```
        .LOAD FOO
```

clears the workfile and then copies library file "*FOO*" into the workfile. Note that the library file is itself unchanged -- a copy is placed into the workfile.

Files may be catenated together and loaded into the workfile, and only selected portions of the files need be referenced. For example,

```
        .LOAD FOO,GOO 40-45,HOO -30
```

first clears the workfile, then copies all of file "*FOO*" into it, then continues by copying in lines 40 through 45 (inclusive) of file "*GOO*", and finally copies in the first 30 lines of file "*HOO*".

The workfile id is always set to the name of the first file loaded. If all of the files loaded have the same type (or if just a single file is loaded), then the workfile is set to the same type. Otherwise, the workfile type is "*DATA*".

.FORMAT

This command is used to print out a formatted document from a file previously built by the "*P*" command. The name of the file to be printed should follow the command name. If the terminal type (see the "*TERMINAL*" command) is such that the printing must be done in multiple passes, then a request will be made for the character set number to print on this pass. Printing is then initiated.

See Section 3 for full details on the use of this command.

These commands are used to maintain the library and workfile. They apply to entire files, not to individual lines of text.

.CLEAR

This command clears the workfile and sets the workfile id to "*CLEAR FILE*".

.LOAD

This command discards the current contents of the workfile and fills it with a copy of specified library files. For example,

```
.LOAD FOO
```

clears the workfile and then copies library file "*FOO*" into the workfile. Note that the library file is itself unchanged -- a copy is placed into the workfile.

Files may be catenated together and loaded into the workfile, and only selected portions of the files need be referenced. For example,

```
.LOAD FOO,GOO 40-45,HOO -30
```

first clears the workfile, then copies all of file "*FOO*" into it, then continues by copying in lines 40 through 45 (inclusive) of file "*GOO*", and finally copies in the first 30 lines of file "*HOO*".

The workfile id is always set to the name of the first file loaded. If all of the files loaded have the same type (or if just a single file is loaded) then the workfile is set to the same type. Otherwise, the workfile type is "*DATA*".

.SAVE

This command copies the workfile into a library file. The name to be given to the library file is specified with the command. Thus

```
.SAVE FOO
```

saves the workfile into library file "*FOO*". If a library file of that name already exists (and therefore would be destroyed by this command), the message "*PURGE OLD FILE?* " will be printed and the keyboard will unlock for input. Enter "*Y*" or "*YES*" to proceed -- otherwise the library file will not be modified.

If no name is specified with the command, then the workfile id will be used. See the "*LOAD*" and "*WFID*" commands for more details on this.

.WFID

This command interrogates/changes the name attached to the workfile. For example,

```
.LOAD JUNK
11/07/78 08.30.11 JUNK

.WFID
IS JUNK

.WFID FOO
WAS JUNK

.SAVE
11/07/78 08.30.45 FOO 1224 BYTES
```

.FILES

This command prints a list, in alphabetical order, of all the names of library files belonging to the account. This includes files which were not created by DICE, as well as all DICE library files. If desired, the listing may be restricted to files beginning with a certain letter sequence, for example

```
.FILES HO
```

lists only files which begin "*HO...*".

.ZAP

This command deletes a library file. The name(s) of files to be deleted follow the command name, separated by commas. For each file, the filename is printed out for verification, and keyboard input is requested. Type "*YES*" to actually delete the file, otherwise press RETURN. Some examples,

```
     .ZAP JUNK
JUNK? YES
```

```
      .ZAP FOO,GOGO
FOO?
GOGO? YES
```

In the first example the file "*JUNK*" was deleted, while in the second example only "*GOGO*" was deleted.

.*SHARE*

Files created by the editor are normally "private" -- that is, they can be accessed only by the user signed on with the *APL* account number under which they were originally created. This is called the owner account. However, the "*SHARE*" command may be used to alter the access control parameters of a file so as to permit access by selected other users.

Access is regulated either (1) on an account-by-account basis, or (2) by a blanket control which applies to all accounts not explicitly granted access. The latter form is commonly termed the "public access", and is conventionally denoted by account number zero. Thus granting an access right to account 0 grants that access to the public.

Four types of access are provided:

(*N*) No access.
(*L*) *LOAD* only.
(*S*) *LOAD* and *SAVE*.
(*) All possible *APL* access.

Note especially that no mechanism is provided whereby one user may create files under another user's account -- the most that can be delegated is the ability to modify a pre-existing file.

In addition to controlling the type of access, it is also possible to require a user to have knowledge of a passnumber before access to the file is granted. Passnumbers are applicable on an account-by-account basis. Zero is equivalent to no passnumber.

The command to permit sharing is of the form

```
      .SHARE filename account:passnumber(access code letter)
```

where the parameters have the following meanings

filename	The name of the file for which the access parameters are to be changed.
account	The account number for which access is to be changed.

passnumber	The passnumber which the account will be required to use in order to access the file. If none is wanted then omit it and the colon.
access code letter	A letter denoting the type of access being granted, one of "*N*", "*L*", "*S*", or "⋆". If this field is omitted then "*L*" is assumed, in which case the "()" characters should be omitted as well.

The command functions by modifying the access control information currently attached to the file, but only for those accounts explicitly mentioned in the command. If other accounts were previously added to the access parameters of the file then they will still remain after the command is executed. The modified form of the access control information is displayed on the terminal. If no new information appears in the command, then no modifications take place and only the display of the old access information is generated.

Some examples:

```
    .SHARE FOO 12345678
```

permits the *APL* user signed on under account 12345678 to *LOAD* file "*FOO*".

```
    .SHARE FOO 12345678(N)
```

withdraws permission[1] for account 12345678 to access file "*FOO*".

```
    .SHARE FOO 12345678(⋆)
```

permits account 12345678 to do anything to file "*FOO*", including *LOAD*, *SAVE*, and *ZAP*.

```
    .SHARE FOO
```

displays the current status of the access parameters of file "*FOO*", but do not change any of them.

Multiple adjustments to the access parameters of a file may be made with a single *SHARE* command, for example,

```
    .SHARE FOO 12345678(L),12340000,45670000(N)
```

[1]actually, account 12345678 is merely deleted from the access matrix of the file. If public access has also been designated for the file, then account 12345678 will of course still be able to access the file as a member of the public. To fully withdraw access permission, account 12345678 should appear in the access matrix with an access code of zero (0). DICE does not support this option, however.

which permits accounts 12345678 and 12340000 to load the file (note that the "(*L*)" could have been omitted) and withdraws permission for account 45670000 to access the file.

These commands provide the connection between the editor and the OS (batch) environment of the IBM System 370/155.

.RUN

This command submits a file to the Internal Job Reader (IJR) of *APL*. This is equivalent to submitting a batch Job to the IBM System/370 via punched cards and a card reader, where each line of the file corresponds to a punched card. If a complete Job stream is submitted, then a Job number will be printed. If a JCL error is found then the message "*FAILED*" will be given. If a partial Job stream is submitted then "*READY*" will be printed. In the latter case, additional "cards" may be submitted via a subsequent "*RUN*" command--as many as are desired. The IJR facility catenates such partial Job streams until a complete Job is recognized.

Some modifications to the JOB card may be done by this command. The cases are:

a) If no Jobname appears on the JOB card, then one is formed from the current workfile id (see the "*WFID*" command).

b) If the character ⍠ appears on the JOB card, it will be replaced by the user's batch password (see the "*PSWD*" command).

Note that these changes are temporary and do not actually appear in the file. Only the submitted Job is affected.

.PURGE

This command is used to cancel a partial job stream which has been submitted to the Internal Job Reader facility of APL via the "*RUN*" command. If successful the message "*PURGED*" is printed, otherwise the message is "*CLEAR*".

.CLOSE

This command submits a single "//" card to the Internal Job Reader facility of APL. It is used to end a Job stream. See the "*RUN*" command for more information on the IJR facility.

Job Card Facility

The following commands may be used to create and edit JOB cards. If the first line of the workfile is not a well-formed JOB card when one of these commands is executed, then a generated card JOB card is inserted into the workfile. It is of the form

```
      // JOB (acctnum,⌹,30s),programmer
```

where "acctnum" is the user's *APL* sign-on number, and "programmer" is the name supplied in answer to the question put by the editor at initialization time concerning the name to appear on JOB cards. See the *RUN* command for some additional discussion of JOB cards.

.*PSWD*

This command is used to change the password (department code) to be used on JOB cards filled in by the "*RUN*" command. A blot will be printed and then the new password is entered. No mechanism is provided to interrogate the present password.

.*TIME*

This command interrogates or changes the time estimate field of the JOB card image which is the first line of the workfile. For example, if the first line of the workfile is

```
//HOMER JOB (12345678,PSWD,30S,200),MYNAME,REGION=120K
```

then the response to the command

```
      .TIME
```

will be "*IS* 30*S*". If a new time estimate is used with the command, for example

```
      .TIME 40S
```

then the response will be "*WAS* 30*S*", and the first line of the workfile will now look like this:

```
//HOMER JOB (12345678,PSWD,40S,200),MYNAME,REGION=120K
```

Similar commands are available to modify other fields of the JOB card, as follows:

COMMAND	MODIFIES
CARDS	Punched card output estimate.
CLASS	CLASS= parameter.
JOBNAME	Jobname.
LINES	Estimate of lines of printed output.
NAME	Programmer name.
PLOT	Estimate of plotter output.
REGION	REGION= parameter.

TYPED FILES

Normally the DICE editor treats the contents of files as collections of completely arbitrary text, the exact content and nature of which is totally ignored by the editor. This can be very frustrating. Consider the following lines of assembler text (line numbers are included, as though this output were produced by the "*p*" command):

```
1>            L      R3,CPBTRCNT
2>            LA     R1,1(R3)
3>            CR     R4,R3
```

Now suppose line 2 were edited, so as to include the label "*STEP*", using Super Edit (*SE*). The following lines might appear on the terminal.

```
.SE
           LA     R1,1(R3)
STEP
STEP             LA     R1,1(R3)
```

The workfile now looks like this:

```
1>            L      R3,CPBTRCNT
2>STEP              LA     R1,1(R3)
3>            CR     R4,R3
```

The original field alignment of line 2 has clearly been lost. It is possible to overcome this problem, chiefly by always using the "Replace" command or its equivalent. For example, the change as done could have been carried out as

```
.RC1+4/STEP/
```

which would have replaced the first four blanks in line 2 with the string "*STEP*". This of course retains the column alignment.

Attempting to solve such problems by using only a subset of the commands considerably reduces the utility of the editor. This problem is addressed by <u>typed files</u>. The user indicates the type of file being edited, and the editor responds by suitably altering the behavior of the editing commands.

Support is planned for these types:

Assembler	(*ASM*)
Data	(*DATA*)
Fortran	(*FORT*)
PL/1, Algol	(*PL/1*)
Text	(*TEXT*)

Files of type "*DATA*" cause the editor to ignore the specific file contents; that is, the commands all work precisely as indicated in their descriptions in the rest of this manual. Currently all other file types except "*ASM*" behave as though they were "*DATA*".

ASM TYPE FILES

Two distinct changes in editing are introduced when the file type is *ASM*. One involves the editing of text already existing within a file (that is, modifications to lines of the workfile), the other concerns the input of new text.

Modifications are done in such a way that column alignment is maintained for the following fields as indicated.

Label - column 1
Operation Code - column 10
Operand Field - column 16
Comment Field - user specified, defaults to 32
Continuation Punch - the character "+" is recognized as a continuation punch and will be held in column 72.
Sequence Numbers - column 73

If a field is long enough to run into the next, then that latter field is moved to the right, just enough to leave a single blank between them. If any field is moved into column 72 or beyond, the behavior of the editor may be unpleasant.

Input is aligned similarly to modified text. A single blank between fields is sufficient to distinguish them -- the editor is then able to align them. If the command extends past column 71, then it is resumed in column 33 of the following card, which also contains an asterisk (*) in columns 1 and 32. If the operand extends past column 71, then a continuation character (+) is placed into column 72 and the field is resumed in column 16 of the following card.

During either input or editing, if the last 8 characters of a line are digits, they are regarded as a sequence number and are moved to columns 73-80, regardless of where they appear in the input, or where they are moved to during editing.

Similarly, if the last character in a line (except for a possible sequence number) is the character "+", then that character is regarded as a continuation punch, and is moved into column 72.

If the operation code is omitted and the operand contains an equals (=) sign and ends with a comma (,) then it will be aligned in column 16, since it is assumed to be a macro keyword parameter.

Commands are provided to interrogate and set the file type.

.TYPE

This command prints out the current type of the workfile.

```
     .TYPE ASM.36
```

This command sets the workfile type to "*ASM*" and sets comment alignment to column 36. If the default column alignment is desired, the ".36" may be omitted.

.TYPES

This command displays the possible file types. See the "*TYPE*" command for details on specific types. For example,

```
     .TYPES
DATA
TEXT
ASM
FORT
PL1
```

.GANG

This command controls "gang punching" on type *ASM* files (see the "*TYPE*" command). A <u>gang punch</u> is a sequence of characters which appears in the same way on each card of a deck. Specifically with DICE, it is a 3 or 7 character sequence designating the name (initials) of the person preparing the input, plus (optionally) the date. For example, for the author, "DEC" and "DEC1178" are both possible gang punches. These strings will be placed on each statement of an Assembler source deck if the gang option has been selected. The command

```
     .GANG XXX
```

will initiate gang punching. Following such a command, further use of gang punching during the editing session may be accomplished by using commands of the form

```
     .GANG OFF
     .GANG ON
```

The editor will remember the initials ("*XXX*" in this case) while gang punching is "off". To interrogate the current state of gang punching, use the command in the form

```
     .GANG
```

The response will be "*IS OFF*" or "*IS XXX*1178".

All of the above applies only to a particular DICE session. If use of the gang punching feature on a regular basis is anticipated, then the gang punching default should be set to the user's initials. This is done by doing a normal activation of gang punching (as noted above) with the word "*DEFAULT*" appended. For example,

```
.GANG XXX DEFAULT
```

Subsequently just "*GANG ON*" and "*GANG OFF*" will suffice to control gang punching. The correct date will automatically be supplied.

These commands provide control over the terminal during a DICE session.

.BELL

This command sets the state of the DICE "bell prompt". If this prompt is "on", DICE will activate the audible alarm (bell) in the terminal whenever it is expecting input. The following forms of the command are valid:

```
    .BELL
```

Queries the current state of the bell prompt. Possible DICE replies are "*IS ON*" and "*IS OFF*".

```
    .BELL ON
    .BELL OFF
```

These forms of the command turn the bell prompt on/off.

When the user signs on to DICE, the bell prompt is set to a default value, normally off. This default value may be interrogated/changed by adding the word "*DEFAULT*" after any of the above forms of the command. Thus these forms are valid:

```
    .BELL DEFAULT
    .BELL ON DEFAULT
    .BELL OFF DEFAULT
```

.HOLD

This command causes the keyboard to unlock for input at the left margin. Press "RETURN" to end the command. DICE then continues normal operation. This is primarily used to permit paper to be inserted/removed from the terminal in the midst of other DICE operations, or to cause DICE to pause so that the paper can be positioned by hand.

.WAIT

This command is exactly equivalent to the "*HOLD*" command.

.TABS

This command sets the APL tabs, and is equivalent to the *APL* system command)*TABS*. For example,

```
      .TABS 6
      WAS 0

      .TABS
      IS 6
```

.TERMINAL

This command is used to advise DICE of the type of terminal being used for this session. The types presently supported are

```
      LA36
      DIABLO
```

When DICE begins running it assumes the terminal type is *LA*36. At the present time it is only necessary to change the type if the formatting subsystem is being used. The terminal type is set by (for example)

```
      .TERMINAL DIABLO
      WAS LA36
```

If no terminal type is specified, DICE indicates the type presently assigned. For example,

```
      .TERMINAL
      IS DIABLO
```

These commands control initiation and termination of DICE sessions.

DICE is started by first signing on to *APL* and then typing

```
    )LOAD 2 DICE
```

DICE will automatically begin execution.

.*OFF*

This command terminates the DICE editing session and returns control of the terminal to *APL*. If it is desired to also sign off *APL*, use the ")*OFF*" form of the command.

If the workfile was saved (and was not subsequently altered) before signing off then it is cleared automatically during the signoff process. Otherwise, the message

```
            "SCRATCH WORKFILE? "
```

is printed, and keyboard input is requested. Replying "*YES*" (or just "*Y*") causes the workfile to be cleared, otherwise the workfile is preserved and will still be present at the next invocation of DICE. Note that with the "*CONTINUE*" command some of this is done automatically.

A command may be specified, as an operand to the *OFF* command, which is to be executed the next time the user signs on to DICE. For example,

```
    .OFF /*LOOK AT FILE HOHOHO BEFORE PROCEEDING/
```

causes the message '*LOOK AT FILE HOHOHO BEFORE PROCEEDING*' to print at the next signon to DICE (see the * command).

.)*OFF*

This command is similar to the "*OFF*" command, but it also signs the user off *APL*.

.CONTINUE

This command is used to sign off DICE and *APL* in such a way that the next sign-on to *APL* will cause DICE to be restarted and the current session to be continued, just as if no sign-off had been performed.

The next time the user signs on to *APL*, the DICE workspace will automatically be loaded. Type "→6" to restart the editor, then continue as though no interruption had occurred.

These commands may prove useful for users with a knowledge of *APL* who wish to use that knowledge in creating and manipulating files.

.*APL*

This command is used to execute *APL* language expressions while remaining under control of DICE. The operand is passed to the *APL* ⍎ primitive. If the expression generates a result, it will be printed. For example,

```
.APL ○1
3.141592654
```

The expression is evaluated in the DICE workspace. Note that the index origin is zero (0). If an error occurs in the executed expression such that suspension occurs, typing →3 will generally restart DICE.

.*TEXT*

This command is used to move text from the *APL* environment into the workfile. The keyboard unlocks under *APL* ⎕ input. Text entered will be taken as material for insertion into the workfile following the current line. If the input is an *APL* vector, then carriage returns embedded in the text are taken as new line delimiters. If the input is an *APL* matrix, then each row thereof is considered to be a line.

For example, suppose the *APL* variable *MYTEXT* has been copied into the editor workspace. Then by using this command, and entering *MYTEXT* in response to the ⎕, the contents of the variable are inserted into the workfile, following the current line. Or, suppose that a table is to be inserted which is of such a form that ∆*FMT* could be used to generate it. Then an *APL* expression using ∆*FMT* could be entered in response to the ⎕. For example, one can easily generate a series of card images containing the integers in sequence from 1 to 100 in the first 10 columns by this means. This might be useful as test data for a program. In response to the ⎕ one enters

```
,('I10' ∆FMT 1+⍳100),CR
```

to accomplish this. Note that the editor workspace is in zero origin and that there is a variable called *CR* which contains a carriage return character.

Another potential use for this command is in selectively copying a known component of an *APL* file into the workfile. Suppose that component 3 of file '123456 *JUNK*' is a complete piece of text which is to be edited. The following sequence of commands accomplishes this. Note that in the display below the hypothetical editor responses are shown at the appropriate places.

```
.⎕
Δ18[5]

      '123456 JUNK' F̲F̲ 100
8
      →5
.TEXT
⎕:
      F̲F̲ 6 8 3
```

Note that the function *F̲F̲* is in the editor workspace, but not the *APL* functions such as *FTIE* and *FREAD* which are available in the workspace 18 *FILES*.

.⎕

This command suspends the DICE editor, in the *APL* sense of the word "suspend", returning the user to calculator mode. DICE may be restarted without loss of continuity by entering →5 .

For example,

```
.⎕
Δ18[5]     (printed by DICE)
           (terminal is now in calculator mode)
      →5   (this restarts DICE)
```

.HELP

This command prints out documentation for a specific command. Actually, this document (LA16) is largely an assembly of the text printed by the *HELP* command. Thus,

```
      .HELP SAVE
```

will print out information on the "*SAVE*" command.

.DO

This command causes DICE to take command input from a file, rather than from the terminal. For example,

```
      .DO FOO
```

specifies that DICE is to read successive lines of input from file "*FOO*" until that file is exhausted, at which point the terminal will once again become the input device. Note that only command input is taken from the file. Other input continues to be taken from the keyboard. In particular, if input of full lines of text is desired, consider using the "*I*" command.

The dister character should not be included in the file. The editor will automatically recognize each line of the file as representing a command.

The "*DO*" command will create a file called "*ΔDICEΔDO*". This file may be erased if desired, but in general it may be ignored.

---

*.**

This command prints out a message on the terminal. The text to be printed must follow the "*". For example,

```
      .*HOW NOW BROWN COW
HOW NOW BROWN COW
      .* HOW NOW BROWN COW
 HOW NOW BROWN COW
```

This command may prove useful when used in a "*DO*" file, in conjunction with the "*WAIT*" command.

.COMMANDS

Prints a list (at the terminal) of all DICE commands.

A detailed list of all commands, including their syntax classifications, is obtained by the command

```
        .COMMANDS LONG
```

---

*.LA*16

Assists in the construction the CCIS Memo "LA16". It functions like the *LOAD* command with this extension: when the sequence "*.INCLUDE* command" is found in the file, the text loaded into the workfile is the "*HELP*" text for the command. This assists in "gluing" LA16 together. The workfile id is "*FULL*" appended to the name of the file loaded.

.PA

This command is used to reprint Public Announcements. DICE will normally print such announcements during user sign-on (to DICE, not to *APL*). However, all announcements are available in a file and can be printed via this command. Announcements are numbered sequentially, starting from 1. The command

```
        .PA
```

prints a message of the form "*LAST PA NUMBER* n". Use the command in the form

```
        .PA m
```

to print out announcement number "m".

.V

This command prints a message and then unlocks the keyboard for input. If the reply is "*YES*" or "*Y*", DICE continues processing; otherwise, it prints the error message "*USER VERIFY FAILED*" and terminates command processing. This may be used in "*DO*" files to support conditional execution of commands. For example,

```
.V CONTINUE? .C/A/B/
CONTINUE? Y
```

In this case the Change (*C*) command is executed. However, in the following case it is not,

```
.V CONTINUE? .C/A/B/
CONTINUE?
```

and the error message indicated above will print. In this case, all commands on the same line as the "*V*" command and to its right will not be acted upon.

.DISTER

This command is used to change the discriminative character, that is, the character which begins a command. Thus, for example,

```
      .DISTER *
```

changes dister from "." to "⋆" (the *APL* star character). The change is permanent, in the sense that it will survive sign-off, until changed again by use of the command.

.=

This command is precisely equivalent to the "*DISTER*" command.

.COST

This command prints the cost of the *APL* session, so far. This includes DICE usage plus any other *APL* usage in this session.

.\

This command executes a user-written *APL* function within the DICE workspace. See Appendix A for details on writing and using such functions.

The DICE Text Formatter is a subsystem of the DICE editor which may be used to prepare documents using either Digital DECwriter-II or Diablo 1620 terminals.

Capabilities

The Text Formatter provides these features:

Paragraphing

Paragraphs may be indented or may be separated by blank lines, with or without indentation of the first line.

Tables

Provision is made for vertical alignment of multiple lines of text, thus permitting inclusion of tabular material.

Justification

Justification (the maintenance of a uniform left and right margin) is supported on the Diablo 1620 terminal only.

Headings and Footings

A specified phrase may be printed at the top or bottom (or both) of each page throughout a document.

Centering

Automatic centering of text is provided where specified.

Overstrikes

Arbitrary binary overstriking is provided on a character-by-character basis. Underlining of text is also supported.

Page Numbering

Automatic page numbering is provided, with user-specified numbering and positioning on the page.

Concordance

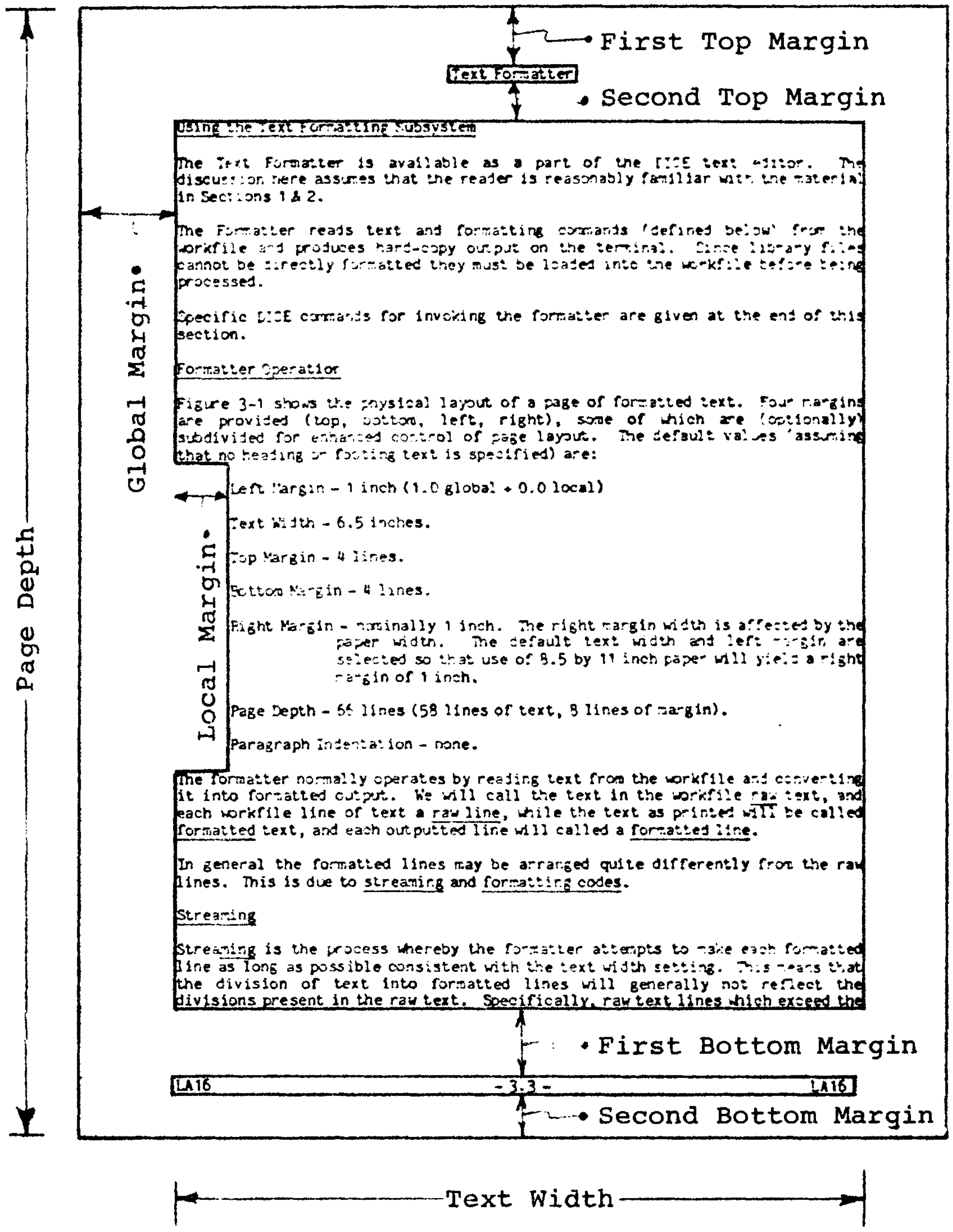
An automatic concordance feature is provided which generates a list of all words used in a document. This may be used to easily detect spelling errors. This is not a part per se of the Formatting Subsystem, but may be used in conjunction with it.

Multiple Character Sets

Support is provided for preparation of documents in mixed typefonts.

Documents using the ASCII and APL characters may be formatted on either the DECwriter-II or Diablo 1620 terminals. On the former the Formatting Subsystem automatically switches the terminal between character sets, so that an entire document may be formatted in a single pass. When using the Diablo 1620 terminal, the document must be passed through the terminal once for each typefont used. No changing of typewheels is done in the middle of a document. With the Diablo 1620, as many typefonts as are desired may be employed, limited only by the number of typewheels available.

For characters not available on standard typewheels, a provision is made for "hand insertion" of text. Blanks are inserted into the document and a mark is placed in the margin to alert the editor to the necessity for hand insertion.



Text Formatter

Using the Text Formatting Subsystem

The Text Formatter is available as a part of the DICE text editor. The discussion here assumes that the reader is reasonably familiar with the material in Sections 1 & 2.

The Formatter reads text and formatting commands (defined below) from the workfile and produces hard-copy output on the terminal. Since library files cannot be directly formatted they must be loaded into the workfile before being processed.

Specific DICE commands for invoking the formatter are given at the end of this section.

Formatter Operation

Figure 3-1 shows the physical layout of a page of formatted text. Four margins are provided (top, bottom, left, right), some of which are (optionally) subdivided for enhanced control of page layout. The default values (assuming that no heading or footing text is specified) are:

Left Margin - 1 inch (1.0 global + 0.0 local)

Text Width - 6.5 inches.

Top Margin - 4 lines.

Bottom Margin - 4 lines.

Right Margin - nominally 1 inch. The right margin width is affected by the paper width. The default text width and left margin are selected so that use of 8.5 by 11 inch paper will yield a right margin of 1 inch.

Page Depth - 66 lines (58 lines of text, 8 lines of margin).

Paragraph Indentation - none.

The formatter normally operates by reading text from the workfile and converting it into formatted output. We will call the text in the workfile raw text, and each workfile line of text a raw line, while the text as printed will be called formatted text, and each outputted line will called a formatted line.

In general the formatted lines may be arranged quite differently from the raw lines. This is due to streaming and formatting codes.

Streaming

Streaming is the process whereby the formatter attempts to make each formatted line as long as possible consistent with the text width setting. This means that the division of text into formatted lines will generally not reflect the divisions present in the raw text. Specifically, raw text lines which exceed the

LA16 - 3.3 - LA16

Figure 3-1

Using the Text Formatting Subsystem

The Text Formatter is available as a part of the DICE text editor. The discussion here assumes that the reader is reasonably familiar with the material in Sections 1 & 2.

The Formatter reads text and formatting commands (defined below) from the workfile and produces hard-copy output on the terminal. Since library files cannot be directly formatted, they must be loaded into the workfile before being processed.

Specific DICE commands for invoking the formatter are given at the end of this section.

Formatter Operation

Figure 3-1 shows the physical layout of a page of formatted text. Four margins are provided (top, bottom, left, right), some of which are (optionally) subdivided for enhanced control of page layout. The default values (assuming that no heading or footing text is specified) are:

Left Margin - 1 inch (1.0 global + 0.0 local)

Text Width - 6.5 inches.

Top Margin - 4 lines.

Bottom Margin - 4 lines.

Right Margin - nominally 1 inch. The right margin width is affected by the paper width. The default text width and left margin are selected so that use of 8.5 by 11 inch paper will yield a right margin of 1 inch.

Page Depth - 66 lines (58 lines of text, 8 lines of margin).

Paragraph Indentation - none.

The formatter normally operates by reading text from the workfile and converting it into formatted output. The text in the workfile will be called _raw_ text, and each workfile line of text a _raw line_. The text as printed will be called _formatted_ text, and each line printed will called a _formatted line_.

In general the formatted lines may be arranged quite differently from the raw lines. This is due to _streaming_ and _formatting codes_.

Streaming

Streaming is the process whereby the formatter attempts to make each formatted line as long as possible, consistent with the text width setting. This means that the division of text into formatted lines will generally not reflect the divisions present in the raw text. Specifically, raw text lines which exceed the

length of a formatted line will be broken up into two or more lines of output. Conversely, short raw lines will be combined into longer formatted lines if necessary.

For example, consider the following raw text:

```
"Fourscore and seven years ago our fathers
brought forth on
this continent a new nation
, conceived in liberty, and dedicated to the
proposition that all men are created equal.[1]
```

Under the default formatting values indicated earlier, this text will be formatted as follows:

```
"Fourscore and seven years ago our fathers brought forth on this continent a new
nation, conceived in liberty, and dedicated to the proposition that all men are
created equal.
```

[1] Some Lincoln scholars insist that Lincoln actually said "upon this continent". Others are vehement in defense of "on". The author, not being a Lincoln scholar, insists upon (or, "insists on") being left out of the swirl of this debate.

Sample Raw Text

```
Zh Zc Text Formatter
Zp1 Zf LA16Zm30 - 3.Zp  -Zm60 LA16

The DICE Text Formatter is a subsystem of the DICE editor
which may be used to prepare documents using either
Digital DECwriter-II or Diablo 1620 terminals.

ZF Capabilities

The Text Formatter provides these features:

Zm5 ZF Paragraphing

Zm10 Paragraphs may be indented or may be separated by blank lines, with
or without indentation of the first line.

Zm5 ZF Tables

Zm10 Provision is made for vertical aligrment of multiple lines
of text, thus permitting inclusion of tabular material.

Zm5 ZF Justification

Zm10 Justification (the maintenance of a uniform left and right margin)
is supported on the Diablo 1620 terminal only.

Zm5 ZF Headings and Footings

Zm10 A specified phrase may be printed at the top
or bottom (or both) of each page throughout a document.

Zm5 ZF Centering

Zm10 Automatic centering of text is provided where specified.

Zm5 ZF Overstrikes

Zm10 Arbitrary binary overstriking is provided on a character-by-character
basis.  Underlining of text is also supported.
```

Figure 3-2

Formatting Codes

Certain sequences of characters in the workfile are not regarded as text, but rather as directions to the formatter specifying special actions to be taken when outputting the remainder of the text. Such character sequences are called formatting codes. These are detailed below. Note the distinction between DICE commands and formatting codes: a DICE command begins with dister, and is acted upon when it is typed in, whereas a formatting code begins with some other character (normally "Z") and is placed into the workfile like any other text. Formatting codes are only recognized by the formatter during formatted output operations.

A sample of the raw text from which part of this document was formatted is shown in Figure 3-2. It illustrates both streaming and the use of formatting codes.

A formatting code is a sequence of three or more characters in the raw text which are interpreted as a command to the formatter, causing it to change the way in which the formatted text is printed. Formatting code sequences never appear in the formatted output.

The three-character sequence always consists of

1. The letter "Z" (or some other character specified by the user), which is called the "formatting code recognition character".

2. A formatting code character (usually a lowercase letter) which defines the function of the formatting code.

3. A single blank.

In addition, some formatting codes include one or more numbers as parameters. These appear after the formatting code letter and before the blank. If multiple numbers are included, they should be separated by a comma (,).

In the definitions given in the remainder of this document, the letter "Z" will always be used for the formatting code recognition character, but the trailing blank will not be explicitly indicated.

The APL characters "α" and "ω" will be used to denote the optional numeric parameters in formatting codes. Thus, if a formatting code is indicated as being of the form "Zaα,ω", then the following are all valid formatting codes:

```
Za
Za4
Za0
Za4,5
Za,5
```

Margin-setting Formatting Codes

Ztα,ω Set first top-of-page margin to "α" lines, second top-of-page margin to "ω" lines. If "ω" is omitted the second margin value is unchanged from its previous value.

Examples

Zt0,0 - eliminate both top margins.

Zt10 - change first top margin to 10.

The formatter uses a default of Zt4,2.

Zbα,ω Similar to the "t" code above, but affects the bottom margins. The first bottom margin is set to "ω" and the second is set to "α". Note that this is reversed (in a sense) from the "t" code. The first number in either code sets the margin which is closest to the physical edge of the paper, and the second number sets the margin which is between the page of text and the heading or footing.

The formatter uses a default of Zb2,4.

Zdα Sets the page depth to "α" lines. The depth may be changed at any point within a document.

Example

Zd33 Set lines per page to 33. This would be used if double-spacing were being achieved by using a terminal with local double-spacing, where in effect each page has only half as many lines as it would have otherwise. Note that the normal page depth is 66 lines. When double-spacing is obtained by use of the "l" formatting code (see "Formatting Codes for Line Control"), the page depth should be left at 66.

The formatter uses a default of Zd66.

Zwα Set text width to "α" tenths of an inch.

Example
Zw60 Set width of a line to 6.0 inches.

Note that if output is being done at 10 pitch (10 character per inch) then the parameter to this code also may be regarded as specifying the number of characters per formatted line.

The formatter uses a default of Zw65.

Zmα Set local left margin to "α" tenths of an inch. This applies to the text which follows the formatting code. Thus, for example,

```
Zm20 abcdef...
```

directs the editor to position "abcdef..." so that the letter "a" is exactly 2.0 inches from the global left margin.

The formatter uses a default of Zm0.

Zmα,ω When the "m" formatting code is used with two margin numbers, "α" has the meaning defined above while "ω" denotes the local margin to use on all <u>subsequent</u> lines, until another "m" or "l" code is encountered.

Example

Zm5,0 This directs the editor to indent 0.5 inches for the text which immediately follows the formatting code, but not indent any subsequent lines (that is, indent them "zero" inches).

Thus

```
Zm5,0 Now is the time for all good ...
```

will result in this output:

```
      Now is the time for all good men to come to the aid of their
country.
```

and

```
Zm5,8 Now is the time ...
```

will result in

```
Now is the time for
   all good men to come
   to the aid of their
   country.
```

Note: see the "l" and "g" formatting codes for some additional information about the "m" formatting code.

Formatting Codes for Line Control

Zl This code directs the formatter to force the raw text which immediately follows to be placed on a new formatted line.

If the formatter has already printed a partial formatted line (carriage is not at the left margin), a carriage return-line feed sequence is generated to return the carriage to the left margin.

Indentation is then done as specified by the "α" value of the most recent "m" code; then formatting is resumed with the raw text which follows the "l" formatting code.

It is important to note that the carriage return-line feed sequence is only done once, and then only if the carriage is not at the left margin. Putting two "l" codes in succession in the document does not cause an extra blank line to appear.

Zg This code differs from the "l" code only in that the indentation is determined by the "ω" value of the most recent "m" code.

Zlα This code directs the editor to put "α" line feeds after each carriage return. Thus 1 means single spacing, 2 double spacing, etc. The default is single spacing.

Zr This formatting code causes the formatter to enter "raw" made. This means that each raw line is treated as though it begins with a "Zl" formatting code.

Zs Resets the formatter to "stream" (normal) mode, i. e., it cancels the effect of a previous "r" formatting code. The default mode is streaming.

Ze This formatting code causes the formatter to begin a new page of formatted output, filling out the current page with blank lines as necessary. The page number is incremented by 1.

Zeα Like "Ze", but the page number is incremented by "α". This can be used when several pages of additional material will be bound into the final document.

Note: When the formatter is positioned at the start of a page, either by coincidence or because of a previous Ze, a Ze formatting code will be ignored. However, if an increment of the page number has been specified, such incrementing will still be done.

Zxα Begin a new page of formatted output, if fewer than "α" lines of output can be placed onto the current output page.

Ziα Insert "α" blank lines into the document. This is in addition to any other blank lines, such as might occur if double spacing were in effect.

Heading and Footing Formatting Codes

As shown on the page layout diagram, a heading line may be printed at the top of each page and a footing may be printed at the bottom.

The text to be used in a heading is specified by the "h" formatting code ("f" for the footing). When either of these codes is encountered in the raw text, the remainder of the raw text line is taken as the heading (footing) text.

Note that these formatting codes do not themselves cause any output to be generated -- they simply cause the text to be stored away within the formatter until the next heading or footing needs to be printed.

Whenever the formatter finishes printing the first top-of-page margin, it checks its internal storage for the presence of a header text phrase. If one is found, it is then formatted and printed. The heading text is printed with these rules:

1. The formatting code recognition character is "Z", regardless of what it may have been changed to elsewhere in the document.
2. The text width is the same as that in effect when the last line of the previous page was printed.
3. The character set in effect is ASCII (character set zero).
4. The character set pitch is the same as that in effect at the end of the last line of output on the previous page.
5. The local margin is 0 (i. e., as though Zm0,0 had been entered).
6. Justification is "on" (i. e., as though Zj had been entered).
7. Single-spacing between lines is in effect.

A similar action is taken by the formatter with respect to the footing code just after the first bottom-of-page margin has been printed.

All formatting codes are acceptable within a heading or footing text and will be acted upon as though the heading or footing text had just been encountered in the workfile. However, all formatter values (margin settings, character set, line spacing, etc.) changed during processing of a heading or footing are restored to their previous values after the heading or footing has been printed.

Zh	This formatting code causes the remainder of the raw text line to be stored for use as the heading at the top of the next formatted page of output.
Zf	Same, except the text is used for the next footing.

Multiple Typefont Formatting Codes

When text is entered using more than one typefont, special steps must be taken to insure that the formatter properly outputs the text. Typefonts are numbered starting from zero, with no particular maximum value specified. Typefonts are identified to the formatter by specifying their associated number. The formatter assumes that the ASCII (upper and lower case) font will always be character set 0 and APL will be character set 2.

Because of the way in which terminals encode characters, DICE cannot determine the character set in use when text is typed into a file. Thus, for example, DICE cannot detect which typewheel is mounted on a Diablo 1620, nor can it determine the character set in which a DECwriter is presently operating.

This principle must be clearly understood when text in multiple fonts is being entered. If a file is entered on a DECwriter operating in the APL character set and the resulting file is then processed by the formatter using the default character set (which is ASCII or character set 0), the output will be ASCII text, not APL text. The characters actually printed are selected by a process known as "keyboard pairing". For a given APL character which was entered, the ASCII character which will be printed during output is the one which is "paired" on the DECwriter keyboard (appears on the same key cap) with the APL character. Thus, every "α" entered will print back as "A", every "*A*" as "a", etc. Because of this phenomenon, phrases which imply that text was "entered in ASCII" or "entered in APL" will be strictly avoided here. The only relevant considerations when entering text are (1) which character set the formatter will use when printing out the text, and (2) the proper character to enter to cause the desired character to print when the document is formatted.

The general rules of entering text are

1. Always operate the terminal in "Escape-2" mode. This is done by entering an Escape character (ESC key on most terminals) followed by the digit 2, which only needs to be done once per session normally. Alternatively, by signing on to APL using the yellow character set on the DECwriter, Escape-2 mode will automatically be selected. Never change the DECwriter character set by placing the terminal in ESC-0 or ESC-1 mode.

2. Advise the editor of the character set to be used for output by preceeding text with a formatting code consisting of the formatting code recognition character (Z), followed by the number of the typefont. The formatter assumes that all text following such a code is in the same character set, until another character set formatting code is encountered. Thus, for example, to cause this output

   the expression *A*←ιρ*X* yields

   (which contains text in character set 0 and 2), enter the raw text as

   the expression Z2 a[IRx Z0 yields

In the above example <u>no APL characters were actually entered into the raw text</u>. Instead, all entry was in ASCII, and the characters "a[IRX" were chosen (by the keyboard pairing rule) so as to cause the proper APL characters to print during output. If this procedure is found to be disconcerting, the "CHAR SET LOCK" button on the DECwriter may be employed (while entering text in character set 2 only) to cause the terminal to print the "correct" characters. Note, however, that using this key has no impact whatsoever on the text which is entered into the workfile, and is not a substitute for entering the formatting codes which select the output character set.

Zα Text following this formatting code will be printed out in the character set numbered "α".

Hand Insertion

Some characters may not be available on any standard typewheel or standard terminal character set. In these cases it may be necessary to insert hand-written text into the final document. DICE supports this by means of a "hand-insertion" formatting code. This code causes blanks to be put into the final document (where the hand-inserted text is to be placed) and a mark to be put in the left margin. After the final copy of the document has been printed, it must be scanned for the margin mark, and each marked line must be hand edited. The purpose of the margin mark is to insure that all hand editing may be done without having to exhaustively scan the final document. The margin marks may be trimmed off or whited out after all hand insertions have been done.

Zqα Leave "α" blank spaces in the document for hand insertion, and put a mark in the margin. If "α" is omitted, a single blank space will be left in the line.

Formatting Codes for Character Control

Z_ This formatting code causes the remainder of the raw text line to be printed out underscored. The underscore character may be entered in either the ASCII or APL character set.

There are two different underscore characters used during output. Text entered in APL will be overstruck with the APL underscore (shift of the letter "*F*"). Text entered in any other character set will be overstruck with the ASCII underscore, which is paired with the "-" symbol. Note that on Diablo typewheels other than ASCII, this character may not in fact be an underscore. It is probably best to avoid underscoring text in other than character set 0 or 2.

Zc This formatting code causes the remainder of the raw text line to be centered. The centered text always occupies one entire formatted line; that is, the centered text appears by itself on a line. The centering is within the space marked "text width" on the page layout diagram.

If the text to be centered cannot fit onto a single formatted line of output, it will be printed as two or more formatted lines, each of which will be centered.

Zo This formatting code causes the next two characters from the raw text to be output as a single, overstruck character. At the present time, the two characters must be in the same character set.

Znα,ω This formatting code directs the editor to adjust the spacing between adjacent characters in character set number "α" such that no more than "ω" characters are printed to the inch. If the editor is not in justification mode (see below), exactly "ω" characters will be printed per inch. This formatting code has no effect upon output done on a DECwriter.

Below are some examples of text formatted at various pitches (characters per horizontal inch).

S i x   c h a r a c t e r s   p e r   i n c h .

E i g h t   c h a r a c t e r s   p e r   i n c h .

Ten characters per inch.

Twelve characters per inch.

Fourteen characters per inch.

Sixteen characters per inch.

The typewheels supplied with the Diablo terminal are designed to be used at either 10 or 12 pitch. These values are only recommendations, however. This document is printed with a font (PICA 10) designed to be used at 10 pitch, but is actually printed at 12. In general, this author prefers to select a pitch which is numerically two greater than that recommended by the typewheel designer. The above examples may be used as a guide to pitch selection.

The pitch of the character set may be changed frequently, even within a word or sentence. Note that this s e n t e n c e contains a word which is printed at 9 pitch. The raw text which generated the preceeding sentence contains this phrase:

```
Note that this Zn0,9 sentenceZn0,12 contains a ...
```

Observe that the blank characters which precede and follow the word "sentence" are both in the normal pitch, that is, the pitch of the rest of the document. When typing text there may be a tendency to naturally enter such a phrase as follows:

```
Note that this Zn0,9 sentence Zn0,12 contains a ...
```

Although this "looks" better since the "Zn0,12" formatting code is set off by blanks, in fact the word "sentence" will have a smaller space before it than will appear after it. This will be disconcerting in many contexts, especially where text in multiple fonts is being produced and the pitch varies from font to font.

The default pitch is 14 for character set zero (0) on a Diablo 1620 terminal. All other character sets on Diablos, and all character sets on DECwriters, are treated at 10 pitch.

Formatting Codes for Justification Control

Zj	This formatting code directs the editor to adjust the spacing between words (and letters if necessary) so that each formatted output line is exactly the length specified by the text width (see the page layout diagram and the "w" formatting code). This formatting code has no effect when output is being done on a DECwriter.
Zk	This formatting code is the opposite of the "j" code above. It directs the formatter to strictly follow the character set pitch specification as indicated by the "n" formatting code, and not to attempt to align the right margin. This formatting code has no effect when output is being done on a DECwriter.

Normally the editor will operate in justification mode. If this is not desired, and the entire document is to be printed non-justified, consideration should be given to using the "TERMINAL LA36" command to DICE , regardless of the terminal type actually being used, since no justification is done on DECwriters. Note that the variable-pitch feature of the Diablo terminal will not be available under such circumstances.

Normally justification should be "off" when entering tabular material in which line-to-line character alignment is needed. Note also that when justification is "on" the blank character is only 0.7 times as wide as the remainder of the character set. This value is integrated into the formatter and cannot be changed. Although this "squeezing" of the blank enhances the appearance of the printed output, it may interfere with the proper representation of certain text. In particular, when quoted strings are used which contain blanks, where the possibility arises that the reader will count the characters in the string, justification should be disabled so that blanks will appear at the same size as the remainder of the string. If this is not done, adjacent blanks may appear to be single blanks.

Justification may be turned "on" and "off" freely within a document, on a character-by-character basis if necessary. When a line contains both justifiable and non-justifiable text, only the spacing of characters in the justifiable portion will vary. If considerable alteration of the line length is needed to align the right margin and the proportion of justifiable text in the line is small, considerable distortion of the text will result. Whether or not this is acceptable will depend upon the judgment of the publisher. To produce reasonable copy under these circumstances, some trial-and-error experiments with justification and character set pitches may be required.

Miscellaneous Formatting Codes

Zp	Whenever this formatting code is encountered in the raw text the formatter injects the current page number, as one or more characters representing a decimal number, into the output. The page number is incremented by 1 for new page of formatted output. Note that the "Ze" formatting code may increment the page number by more than 1.
Zpα	This formatting code sets the page number to "α".
Z=x	This formatting code is used to change the formatting code recognition character ("Z" in these examples). It is changed to the character "x", which may be any printing character (including blank!).
ZZ	This formatting code causes a single "Z" to be written in the formatted output.

Errors in Formatting Codes

When the formatter encounters a character sequence in the raw text which resembles a formatting code, but which is actually invalid, it treats it as normal text. No warning is signaled by the formatter when this happens.

DICE Commands for the Formatting Subsystem

To assist in running the formatter several commands have been added to the DICE Editor (not the formatter). These are described below.

.TERMINAL DIABLO
*.TERMINAL LA*36

> These commands must be used to advise the editor of the type of terminal which is being used, before invoking the formatter. This command need only be entered once per session. The editor assumes the terminal is an LA36 (DECwriter) unless otherwise specified.

A large number of modifiers have been added to the DICE "*P*" command to assist in invoking the formatter. The "*F*" modifier must always be included to invoke the formatter. The "*F*" modifier has already been discussed as being the mechanism whereby the formatter is activated. The additional modifiers are as follows.

*P*α Set the initial page number to α.

*M*α Format the document with a global left margin of "α" tenths of an inch. This value defaults to 5, which is useful for DECwriters which otherwise will print too close to the left-hand edge of the page.

*S*α Print only character set number "α". All other character sets will print as blanks. This is used with the Diablo terminals when paper is being passed repeatedly through the terminal in order to print documents in multiple character sets.

*W*α Print the document with an initial text width of "α" tenths of an inch.

*T*α Print the document with a first top-of-page margin of "α".

*B*α Print the document with a second bottom-of-page margin of "α".

*D*α Print the document with a page depth of "α". The default is 66.

User-written DICE Commands

(Appendix A)

The present version (Version 2) of DICE is written entirely in *APL*. Because of this, it is possible to

(1) interface the DICE workspace to other workspaces, in such a way that DICE can be effectively incorporated into other *APL*-based applications, and

(2) permit users to write *APL* functions to run in the DICE workspace, effectively allowing creation of customized commands.

Linking to DICE

The *AUTOLOAD* facility of *APL* may be used to automatically invoke the DICE workspace. A command line may be set up in advance by loading the appropriate component of the *ΔDICEV2* file. See Appendix C for details on component usage of the file.

When the user "signs off" DICE, control may be returned to the workspace which originally invoked the DICE workspace.

The *DICELINK* function in workspace 2 *DICEHELPER* performs both of these operations. It is designed to be copied into another workspace and used to link to and return from DICE. Comments in the function explain how it is to be invoked.

User-written DICE Commands

Users may define additional DICE commands by writing *APL* functions which implement them. Details on writing such functions are given below. The functions are placed into a file, along with a "dictionary" of the names of the commands they implement, and DICE is informed of the name of the file. The commands are invoked from DICE by entering the command name preceded by a "\" character. Parameters may also be passed to the command function. Details are given below.

The rules for writing a user function to implement a DICE command are:

1. The function must be niladic. Its name must be the same as the command name.

2. The function may call other user-written functions, but the names of all called functions must be known in advance. This restriction applies because DICE must know the names of all other user-written functions to load along with the command function. Note that the keyword *ΔFX* is in the DICE workspace, so the user function may also fix other functions into the workspace.

3. Certain functions and variables in the DICE workspace are available to the user function. Interested users should contact the author for details.

4. In DICE Version 3, Modification 0, there are just over 25000 bytes of

workspace available to the user function. This number is subject to change without prior notification.

Installation of User Functions into the File

All of the user-written functions to be used within DICE must be written into a file known to DICE. Establish a file, and place its name into the appropriate component of the *ΔDICEV2* file (see Appendix C for details), as an *APL* character vector. Component 1 of the file is a directory of the file. It is a character matrix with one row per function, row 1 naming the function which is in component 2 of the file, etc. Each function component is a canonical representation matrix of a function, with an extra row prefaced which is a list of all user-supplied functions called by that function, with semicolon (;) delimiters.

Using User Commands in DICE

The backslash (\) character is used to invoke user commands. When used without an operand, it lists the names of all user-written functions. When followed by the name of a user function, it loads that function into the DICE workspace, along with any called functions, and then executes it.

An Apology

The discussion above is deliberately very sketchy. However, it should serve to give an indication of the capabilities of DICE and *APL*-based extensions thereto. Interested users are urged to contact the author for more information.

Command Syntax

(Appendix B)

In Section 2 each command was presented informally, along with some examples of usage. No complete description of the command syntax was given. In this Appendix a complete syntax description is presented for each command. A modified BNF grammar is used. The angle brackets (<>) are used in their standard sense. Double quotes (") enclose Engligh-language descriptions which are not further explained. Material enclosed in square brackets ([]) is optional.

Below is a list of all DICE commands, with a description of the syntax of each command.

A	*A* [<string>][<modifiers>]
APL	*APL* "any valid *APL* expression"
ASEQ	*ASEQ* <number>[*BY* <number>]
B	*B*
BELL	*BELL* [<bell set>][*DEFAULT*]
	<bell set> := *ON* | *OFF*
C	*C* <double string>[<modifiers>]
CARDS	*CARDS* ["arbitrary text"]
CLASS	*CLASS* ["arbitrary text"]
CLEAR	*CLEAR*
CLOSE	*CLOSE*
COMMANDS	*COMMANDS*
CONTINUE	*CONTINUE*
COPY	*COPY* [<file name list>]
COST	*COST*
D	*D* [<string>][<modifiers>]
DISTER	*DISTER* "any character"
DO	*DO* [<file name list>]
E	*E* [<string>][<modifiers>]

FILES	*FILES* ["arbitrary text"]
FORM	*FORM* <form name>
	<form name> := 80 | *PR*
FORMAT	*FORMAT* <file name>
FORMS	*FORMS*
GANG	*GANG* [<gang set>][*DEFAULT*]
	<gang set> := *ON* | *OFF* | "3-character initials"
HELP	*HELP* | *HELP* ["command name"]
HOLD	*HOLD*
I	*I* [<string>][<modifiers>]
J	*J* [<string>][<modifiers>]
L	*L* [<string>][<modifiers>]
*LA*16	*LA*16 <name>
LINES	*LINES* ["arbitrary text"]
LIST	*LIST* [<file name list>]
LOAD	*LOAD* [<file name list>]
MOVE	*MOVE* <line selector>
NAME	*NAME* ["arbitrary text"]
OFF	*OFF* [<string>]
P	*P* [<string>][<modifiers>]
PA	*PA* [<number>]
PLOT	*PLOT* ["arbitrary text"]
PRINTN	*PRINTN* [<file name list>]
PRINT	*PRINT* [<file name list>]
PSWD	*PSWD*

PUNCH	*PUNCH* [<file name list>]
PURGE	*PURGE*
R	*R* [<string>][<modifiers>]
REGION	*REGION* ["arbitrary text"]
RUN	*RUN* [<file name list>]
S	*S* [<string>][<modifiers>]
SAVE	*SAVE* [<file name>]
SE	*SE*
SEQ	*SEQ* [<seq set>]
	<seq set> := *OFF* | <number> [*BY* <number>]
SHARE	*SHARE* <file name> [<share list>]
	<share list> := <share account> | <share account>,<share list>
	<share account> := <number>[(<share letter>)]
	<share letter> := *L* | *S* | *N* | *
T	*T*
TABS	*TABS*
TERMINAL	*TERMINAL* [<terminal type>]
	<terminal type> := *LA36* | *DIABLO*
TEXT	*TEXT*
TIME	*TIME* ["arbitrary text"]
TYPE	*TYPE* [<file type>]
	<file type> := *ASM* | *DATA* | *FORT* | *PL1* | *TEXT*
TYPES	*TYPES*
U	*U* [<string>][<modifiers>]
V	*V* ["arbitrary text"]

WAIT	*WAIT*
WFID	*WFID* [<file name>]
ZAP	*ZAP* [<file name list>]
?	*?*
⎕	⎕
=	="arbitrary character"
\	\
*	*
)*LOAD*	)*LOAD* "workspace name"
)*OFF*	)*OFF* [<string>]

Below are the descriptions of syntactic classes not defined above.

```
<string> := /"arbitrary text"/ |
             Q<q-character>"arbitrary text not containing
                         the q-character"<q-character>

      <q-character> := "any character"

<double string> := /"arbitrary text"/"arbitrary text"/ |
                      Q<q-character>"arbitrary text"<q-character>
                        "arbitrary text"<q-character>

<modifiers> := <modifier> | <modifier><modifiers>

     <modifier> := <a-modifier letter> |
                   <b-modifier letter>[<number>] |
                   <c-modifier> |
                   <number> |
                   *

     <a-modifier letter> := V | A | N | F

     <b-modifier letter> := L | P | M | S | W | T | B | D | H
```

<c-modifier> := *C*<number>[+<number>] |
C[<number>]-[<number>]

<number> := "one or more decimal digits"

<name> := <letter> | <name><digit> | <name><letter>

<letter> := *A* | *B* | *C* | *D* | *E* | *F* | *G* | *H* | *I* |

J | *K* | *L* | *M* | *N* | *O* | *P* | *Q* | *R* |

S | *T* | *U* | *V* | *W* | *X* | *Y* | *Z* | Δ |

$\underline{\mathit{A}}$ | $\underline{\mathit{B}}$ | $\underline{\mathit{C}}$ | $\underline{\mathit{D}}$ | $\underline{\mathit{E}}$ | $\underline{\mathit{F}}$ | $\underline{\mathit{G}}$ | $\underline{\mathit{H}}$ | $\underline{\mathit{I}}$ |

$\underline{\mathit{J}}$ | $\underline{\mathit{K}}$ | $\underline{\mathit{L}}$ | $\underline{\mathit{M}}$ | $\underline{\mathit{N}}$ | $\underline{\mathit{O}}$ | $\underline{\mathit{P}}$ | $\underline{\mathit{Q}}$ | $\underline{\mathit{R}}$ |

$\underline{\mathit{S}}$ | $\underline{\mathit{T}}$ | $\underline{\mathit{U}}$ | $\underline{\mathit{V}}$ | $\underline{\mathit{W}}$ | $\underline{\mathit{X}}$ | $\underline{\mathit{Y}}$ | $\underline{\mathit{Z}}$ | $\underline{\Delta}$ |

<file name> := [<number>]<name>[:<number>]

<line selector> := <number> | [<number>]-[<number>]

<full file name> := [<file name>][<selector>]

[<file name list>] := <full file name> | <full file name>,<file name list>

Notes on the Syntax

DICE commands fall into four distinct syntactic classes. These are called "string", "filename", "none", and "arbitrary". The names reflect the type of operand each command will accept.

string

Commands in this classification accept text string(s) and modifiers. The command names are all one (1) character long.

A string operand consists of an (optional) character string, normally enclosed by slash (/) characters, plus modifiers. A modifier is a letter, followed in some cases by a number, which is used to change the basic way the command works. Normally a modifier has a similar effect regardless of the command to which it is applied. The modifiers presently supported are:

A	(not used)
**B*	Bottom. Selects bottom margin (for formatted output only).
**C*	Columns. Restricts the action of the command to selected columns.
**D*	Depth. Sets page depth (for formatted output only).
F	Format. Used to print formatted output ("*P*" command only).
**H*	(not used)
**L*	Lines. Causes the command to apply to a range of lines.
**M*	Multiple. Causes the command to apply to multiple occurrences, normally in a given line.
N	Numbers. Causes display of line numbers along with normal line display.
**P*	Print. Causes changed lines to be printed.
Q	Quote. Must precede the first quoting character in a command, when / is not used.
**S*	Character set. Restricts formatted output to a single character set.
**T*	Top. Selects top margin (for formatted output only).
V	Verify. Causes the command to request confirmation before altering the workfile.
**W*	Width. Selects the page width (formatted output only).

Those modifiers marked with a star (*) may optionally be followed by a number.

filename

Commands in this classification work upon library files, and (in some cases) are extended to apply to the workfile. Their operand will consist of a file name, optionally followed by an expression which selects just some range of lines from the file, or a list of such operands, separated by commas.

<u>none</u>

Commands in this classification do not accept any operand.

<u>arbitrary</u>

This is the "catch-all" command syntax classification. Any operand (or none) is permitted.

The ∆*DICEV*3 File

(Appendix C)

The ∆*DICEV*3 file contains the user's workfile, as well as information about the user which makes up the "user profile". The usage of the components of the file are as follows. No understanding of this material is necessary to use DICE.

01	*BATCH PASSWORD FOR ⍠.*
02	*DEFAULT PROGRAMMER NAME FOR JOB CARDS.*
03	*LAST PA NUMBER RECEIVED.*
04	
05	*TYPE OF WORKFILE.*
06	*DISTER.*
07	*WORKFILE COMPONENT DIRECTORY VECTOR (CDV).*
08	*WORKFILE ACCESS MATRIX.*
09	*WORKFILE NAME.*
10	*WORKFILE PASSNUMBER.*
11	*NUMBER OF SIGN-ONS TO DICE.*
12	*COMMAND TO BE EXECUTED AT NEXT INVOCATION OF DICE.*
13	*DEFAULT INITIALS FOR 'GANG'.*
14	*USER FUNCTIONS FILE NAME.*
15	*DEFAULT BELL SETTING. 1 = NOBELL.*
16	
17	
18	*NAME OF WS TO AUTOLOAD AFTER SIGNOFF.*
19	*ACCESS MATRIX FOR NEWLY CREATED FILES.*
20	*DEFAULT SUOPT PSWD.*
21	*OPERAND ALIGNMENT FOR TYPE ASM (DEFAULT=32).*

The remainder of the file is used for storage of workfile text.

Initial Session

(Appendix D)

This Appendix presents a sample initial session of DICE. Of particular interest here are the questions which DICE asks the first time it is run under an *APL* account number.

In what follows, the *APL* character set will be used to represent output from the terminal. The "normal" uppercase/lowercase characters represent typed input. Following the sample session, there is a discussion of the meaning of selected parts of the session.

<u>Sample Initial DICE Session</u>

```
      )LOAD 2 DICE
SAVED 13.46.42 03/13/79
YOU ARE A NEW USER OF VERSION 3 OF DICE
FOR CERTAIN DICE COMMANDS WE WILL ASSUME THAT THERE IS
A BATCH ACCOUNT NUMBER 12345678
ENTER THE PASSWORD FOR THIS ACCOUNT: junk
ENTER THE PROGRAMMER NAME WHICH YOU NORMALLY USE ON YOUR JOB CARD
IF NONE, PRESS  RETURN
smithers
YOU ARE NOW ENROLLED IN DICE!

DICE V3M3
YOUR DISTER IS .
```

Recall that DICE runs as part of the *APL* system on the IBM System 370/155. Hence it is necessary to sign on to *APL* before running DICE. This step is not shown in the above example. Rather, it starts just after the sign-on process is completed. The command ")*LOAD 2 DICE*" directs the *APL* system to begin running DICE. DICE then takes over control of the terminal.

Some operations which DICE can perform require that the user have a <u>batch</u> account number on the 370, as well as an *APL* account. Therefore, DICE asks the questions about the password for the batch account number. If there is no such batch account number, or its password is not known, simply press RETURN when asked for the password.

Certain DICE commands submit Jobs to the 370 (for example, *PRINT* and *PUNCH*, among others). When such commands produce batch output, it is necessary for DICE to supply a "name" to be used on the Job, in order that the batch output may be recovered in the User's area. Therefore, DICE must know the proper name to use in such cases. If DICE facilities which depend upon this information will not be used, simply press RETURN in response to the question.

Once these questions have been answered, DICE will automatically begin running. It types the messages

```
      DICE V3M3
```

YOUR DISTER IS .

and unlocks the keyboard for input. At this point, input may be typed to the workfile, or any of the DICE commands described elsewhere in this document may be used.