AD-762 010

AN EXPERIMENTAL DISPLAY PROGRAMMING LANGUAGE FOR THE PDP-10 COMPUTER

William M. Newman

Utah University

Prepared for:

Rome Air Development Center
Advanced Research Projects Agency

July 1970

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

An experimental display programming language for the PDP-10 computer

WILLIAM M. NEWMAN

UNIVERSITY OF UTAH

AD 762010

DDC
RECEIVED
JUN 25 1973
B

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

JULY 1971
UTEC-CSc-70-104
COMPUTER SCIENCE, UNIVERSITY OF UTAH
SALT LAKE CITY, UTAH 84112

AN EXPERIMENTAL
DISPLAY PROGRAMMING LANGUAGE
FOR THE
PDP-10 COMPUTER

by

William M. Newman

July 1970 UTEC-CSc-70-104

This research was supported in part by the University of Utah Computer Science Division and the Advanced Research Projects Agency of the Department of Defense, monitored by Rome Air Development Center, Griffiss Air Force Base, New York 13440, under contract AF30(602)-4277. ARPA Order No. 829.

TABLE OF CONTENTS

ABSTRACT

An experimental language for display programming, called DIAL, has been developed for the PDP-10 and the UNIVAC 1559 display. It is experimental in the sense that it was originally conceived as a means of testing out some ideas, and the best way to test them seemed to be to produce a language that others could use. The language is a subset of ALGOL (hence the name: Display Algol), with additional facilities for graphical input and output. It cannot deal with floating-point numbers and can only handle strings in a limited fashion. Also, it lacks any facilities for rotating pictures or for displaying three-dimensional objects. On the other hand, it does include features which may make it easier to develop display programs.

The principal distinguishing feature of Dial is the ability to define display procedures. These are identical in almost every respect to ordinary procedures, but serve the additional purpose of defining the structure of the picture on the screen. In this respect they take the place of the traditional structured display file, which in Dial does not exist. The only display file created by Dial programs is a linear list of vectors which is sent to the display.

The chief difference between display procedures and other Dial procedures lies in the way they are called. A typical display procedure call might be:

CAPAC AT 100, 200 SIZE 20;

Display procedures may be defined in terms of basic graphical primitive (lines, points, etc) or by means of calls to other display procedures.

Also included in Dial are statements for defining the interactive processes within the program. Dial does not permit a very high degree of graphical interaction: it is not possible to program operations like drawing rubber-band lines or dragging objects around the screen. It is not clear whether this necessarily means that Dial programs are less efficient interactively; in any case, Dial is designed to function under a time-sharing environment where it is difficult to create continuously changing pictures.

DISPLAY PROCEDURES

All statements which produce graphical output must occur within display procedures, or within procedures called by them.* There are four basic graphical output statements: LINE, LINE TO, MOVE and MOVE TO. LINE and MOVE define relative movements, visible and invisible respectively, while LINE TO and MOVE TO define movements to absolute positions in the coordinate system of the display procedure. This is illustrated by the following example, which defines the capacitor symbol shown in Figure 1:

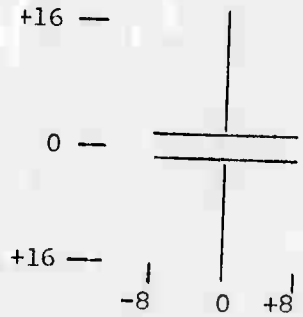


Figure 1

```
CAPAC ← 'MOVE TO 0, -16; LINE TO 0,-2; MOVE -8,0; LINE 16,0,
        MOVE -16,4; LINE 16,0; MOVE -8,0; LINE TO 0,16'
```

In an example of this sort, either relative or absolute movements could be used throughout, instead of the mixture of absolute and relative employed here for the purpose of illustration. Each type of statement finds specific uses in more complex procedures.

Text can be displayed by means of the DISPLAY statement:

```
DISPLAY "ANSWER = ", N AT 100, 200
```

This will display the value of N as a decimal integer, prededed by the

*If graphical output statements are instead included in the main body of the program, they do not constitute syntax errors, but no output will be generated by them.

string "ANSWER = ". This statement is equivalent to the following two statements, which are also permissible in Dial:

```
MOVE TO 100, 200; DISPLAY "ANSWER = ", N;
```

Besides including these basic graphical output statements, display procedures may also call other display procedures. This is illustrated by the next example, which uses the capacitor symbol and another symbol of a resistor to create the circuit shown in Figure 2. The code for this procedure is as follows:

```
CCT ← 'CAPAC AT 32, 24; RESIS AT 0, 40, RESIS AT 32, 40,
      MOVE TO 0, 8; LINE TO 64,8'
```

When we call CAPAC at the position of 32,24 in the outer procedure, this becomes the local origin for the inner procedure. Hence the first MOVE TO 0, -16 in CAPAC will move to 0, -16 in the local coordinate system, which is 32, 8 in the coordinate system of CCT.

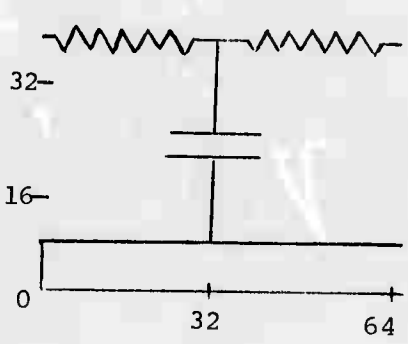


Figure 2: CCT Procedure

The CCT procedure could itself be called from another display procedure, and so on. In this way a hierarchy of procedures can be created, modeling the structure of the picture in much the same way as would a conventional structured display file. There are two fundamental differences, however, between a structure built of display procedures and a similarly structured display file. In the first place, display files are a form of data structure, whose elements may be created, modified and destroyed by the program. Display procedures are defined prior to compilation, and can never be altered during the course of execution.

This raises the question of how pictures can be made to change at all. The answer lies in writing display procedures which make access to the same data structures as the rest of the program, so that when the display procedure is executed it produces a picture reflecting the latest state of the data structure. Consider an example of a more elaborate electrical circuit, containing a number of capacitors. The positions of these capacitors on the circuit diagram can be stored in two arrays, CX and CY, which hold the x and y coordinates respectively. Then a procedure for generating the whole circuit diagram might be written as follows:

```
CCT ← 'FOR K ← 1 STEP 1 UNTIL NC DO CAPAC AT CS[K], CY[Y]'
```

NC is the number of entries in the CS and CY arrays.

Display procedures may have arguments, and this provides another method of altering the picture which the procedure generates. The value of the capacitor could be added as an argument, as follows:

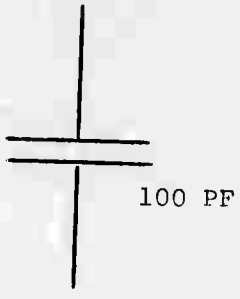


Figure 3

```
CAPAC ← 'NEW V; DISPLAY V, "PF" AT 4,-8;
         MOVE TO 0,-16; etc... '
```

Figure 3 shows the result. Because the procedure is executed by the computer, and not by the display processor, it is possible to include conditional graphical output statements:

```
CAPAC ← 'NEW V; IF LABEL=1 THEN DISPLAY V, "PF" AT 4,-8;
         MOVE TO 0,-16; .... '
```

This procedure will generate the label or omit it according to the value of LABEL.

WINDOWING AND SCALING

The calling of one display procedure by another may be carried to many levels. The result is a pattern of procedure calls resembling a tree structure with cross connections, as shown in Figure 4. As the size of this structure grows, so does the amount of material on the screen, and it soon becomes desirable to be able to select part of the total picture for display. This is achieved by means of the WINDOW statement. Windowing is applied to the display procedure which forms the top node in the tree, using the following syntax:

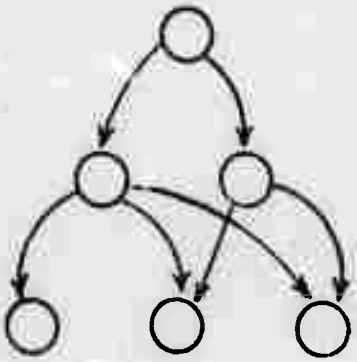


Figure 4

```
WINDOW CCT AT 250, 300 SIZE 200
```

This statement has the effect of placing a square frame around part of the picture defined by CCT. The center of this frame is at 250, 300 in the coordinate system of CCT, and it measures 200 units in each direction from this center. When the WINDOW statement is executed, all the graphical information which lies outside the 'window' is omitted from the picture, and the lines which lie inside are scaled so that the edges of the window coincide with the edges of the display screen.* Lines which cross the boundary of the win-

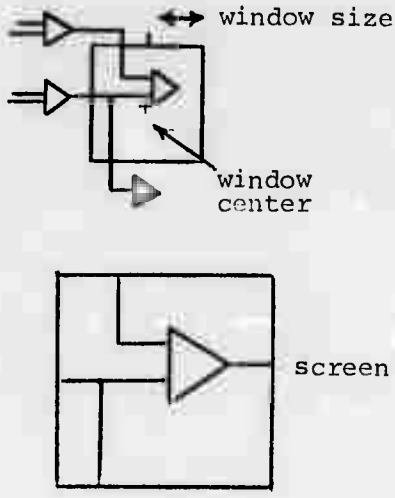


Figure 5: Windowing

*The use of viewports, as proposed by Dr. Ivan Sutherland, is not currently included in Dial.

dow are clipped so that only the visible part is sent to the display. Figure 5 illustrates the effect of windowing.

Windowing can be applied only to entire pictures, and not to individual display procedures which make up part of a picture. This means that scale changes can be applied only to entire pictures by the WINDOW statement. Often one wants to be able to change the scale at which a symbol appears within a picture; this can be done by including the size in the display procedure call:

```
CAPAC AT X1, Y1, SIZE 256;
```

Size is measured in units of the coordinate system of the calling display procedure. This method has been chosen rather than using relative scale, partly because this is often the most convenient way to specify the scale of an object, and partly because the use of relative scales would require floating-point arithmetic. An argument against the use of size is that it implies that every display procedure must be defined at a certain size; and this is in fact the case. The size of each display procedure is included in the declaration of that procedure, in the manner described in the section on declarations. When the procedure is called with an explicit size, as in the last example, it is scaled up in the ratio of the called size to the declared size. If no size is mentioned in the calling statement, the declared size is used.

It is generally immaterial at what size a display procedure is declared, as long as all the graphical information in the procedure is enclosed within the definition boundary, which is assumed to have 0,0 at its center. It is advantageous, however, to make the declared size contain the graphical information as closely as possible. In this way the genera-

tion of display files is greatly speeded up.* As an example, the obvious choice of declared size for the CAPAC symbol would be 16, and 64 for the CCT of Figure 2. The statement on the previous page would then be equivalent to scaling up CAPAC sixteen times.

A very useful feature of Dial is the reserved procedure SSIZE which can be called within any display procedure, and returns the actual size, in grid units, that the procedure will appear on the screen. According to the value returned by SSIZE, parts of the picture may be included or left out, or a completely different picture may be generated. SSIZE could, for example, be used instead of LABEL in the last example on Page 4:

```
CAPAC ← 'NEW V; IF SSIZE> 64 THEN DISPLAY V, "PF" AT 4, -8;
    MOVE TO 0, -16; ...';
```

Any statements or expressions may be used as the coordinates and size in a display procedure call or in a WINDOW statement. The scaling effect of sizes is cumulative when calls are made to more than one level.

---

*Any display procedure which lies entirely outside the current window is omitted entirely without examining its contents. This so-called 'boxing' operation greatly speeds up the clipping process.

FRAMES

Whenever a WINDOW statement is executed, a process of display file generation is begun. The display procedure named in the WINDOW statement is called, and every line or character specified in the ensuing statements is clipped with respect to the window. For each line or character which appears within the window, the appropriate display commands are added to the display file, in preparation for transmitting the file to the display buffer. This process continues until the end of the named display procedure is reached.

Whenever a change is required in the display picture, this process must be repeated in order to regenerate the display file. The new display file must replace the old one in the display buffer. A problem arises if the buffer contains the results of several WINDOW statements, since only the appropriate part of it must be overwritten, and the rest must remain unchanged. To cope with this problem, each separate part of the display file is treated by Dial as a separate frame, and the FRAME statement is included in the language to allow parts of the display file to be regenerated.

The FRAME statement is in reality just a procedure call, and the frame procedure which it calls usually contains just one WINDOW statement:

```
F1 ← 'WINDOW CCT2 AT XW, YW SIZE 1000';
FRAME F1;
```

The FRAME statement carries out two important tasks: it deletes the appropriate parts of the old display file; and when the execution of the frame procedure is complete, it sends the new display file to the display buffer. It is important to realize that only the FRAME statement has

this ability to transmit new display files to the display. The other statements we have discussed are merely concerned with creating a display file in preparation for transmission.

The Dial system keeps a list of all the frames which are currently being displayed, and can in this way keep track of the parts of the display buffer which must be changed when a FRAME statement is executed. If the programmer wishes to remove a frame entirely from the display, he can use the statement:

```
DELETE F1;
```

which also removes this frame's entry from the frame list.

Table I illustrates the effect of a typical sequence of FRAME and DELETE statements, beginning with a clear screen.

Statement	Effect	Display File Contents
FRAME F1	Generate F1	F1
FRAME F2	Generate F2	F1, F2
FRAME F1	Regenerate F1	F1, F2
FRAME F3	Generate F3	F1, F2, F3
DELETE F2	Remove F2	F1, F3
DELETE F2	No effect	F1, F3
DELETE F1	Remove F1	F3
FRAME F3	Regenerate F3	F3
DELETE F3	Remove F3	Empty

Table I: FRAME, DELETE Sequence

DECLARATIONS

Every Dial program must start with a title, a BEGIN and some declarations. Included in the declarations must be the names of all the variables and procedures in the program, except for the formal variables declared at the start of procedures. Variables, arrays, ordinary procedures and frame procedures are declared in NEW declarations:

```
NEW XW, YX, CX[100], CY[100, PROC3, FRAME1, FRAME2;
```

Arrays may have any number of dimensions. Each dimension has an implicit lower bound of 1. Display procedures are declared separately:

```
DISPLAY PROCEDURE CCT, CCT2, 16:  CAPAC, RESIS;
```

A number like '16:' included in the declaration defines the size of the following procedures. A default value of 1024 is assumed for any name not preceded by a size.

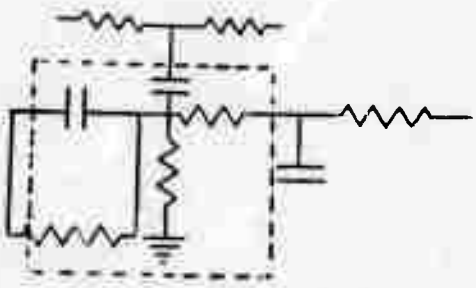
Two other types of declaration, INTERNAL and EXTERNAL, are permitted. These will be discussed more fully in the section on linking to other programs, but their use is probably obvious to those familiar with the PDP-10.

AN EXAMPLE

All the aspects of Dial which relate to picture generation have now been covered, and an example is included here to illustrate how the various Dial statements are used. This example will create the circuit diagram shown below, with the same window. Note the use of a FOR statement to define resistor and ground symbols.

```
TITLE EXAMPLE
BEGIN
NEW VERT,HORIZ,FR1,K;
DISPLAY PROCEDURE CCT, 32:RESIS,CAPAC, 8:GROUND;

CAPAC ← 'NEW HV; IF HV=HORIZ THEN
       BEGIN MOVE 16,Ø; LINE Ø,14; MOVE -8,Ø; LINE 16,Ø;
         MOVE -16,4; LINE 16,Ø; MOVE -8,Ø; LINE Ø,14
       END ELSE
       BEGIN MOVE Ø,16; LINE 14,Ø; MOVE Ø,-8; LINE Ø,16;
         MOVE 4,-16; LINE Ø,16; MOVE Ø,-8; LINE 14,Ø
       END';

RESIS ← 'NEW HV; IF HV=HORIZ THEN
       BEGIN MOVE Ø,16; LINE 4,Ø;
          FOR K←1 STEP 1 UNTIL 3 DO
          BEGIN LINE 2,4; LINE 4,-8; LINE 2,4 END;
          LINE 4,Ø
       END ELSE
       BEGIN MOVE 16,Ø; LINE Ø,4;
          FOR K←1 STEP 1 UNTIL 3 DO
          BEGIN LINE 4,2; LINE -8,4; LINE 4,2 END;
          LINE Ø,4
       END';

GROUND ←  'MOVE 4,2;
         FOR K←Ø STEP 1 UNTIL 4 DO
         BEGIN LINE 2*K,Ø; MOVE -(2*K+1),1 END;
         MOVE TO 4,6; LINE Ø,2';
```

```
CCT ← 'RESIS[HORIZ] AT Ø,Ø;
      MOVE TO Ø,16; LINE Ø,32; MOVE TO 32,16; LINE Ø,32; LINE 12,Ø;
      CAPAC[HORIZ] AT Ø,32;
      CAPAC[VERT] AT 28,48;
      RESIS[VERT] AT 28,16;
      GROUND AT 36,Ø, SIZE 16;
      RESIS[HORIZ] AT 44,32;
      RESIS[HORIZ] AT 76,32;
      CAPAC[VERT] AT 6Ø,16;
      RESIS[HORIZ] AT 12,64;
      RESIS[HORIZ] AT 44,64';

FR1 ← 'WINDOW CCT AT 36,36 SIZE 32';

HORIZ←Ø; VERT←1;
FRAME FR1
END
```



INTERACTION WITH DIAL PROGRAMS

To assist in defining the interactive elements of Dial programs, an ON statement is provided. This statement, which is reminiscent of parts of PL/I, has the form:

ON <input> DO <statement>;

The ON statement is generally executed immediately after input has been received from the Teletype or from the SRI Mouse. If the received input matches the first part of the ON statement, then the statement which forms the second part is executed. The various types of input are as follows:

ON CHAR DO ...	any Teletype character
ON CHAR "Q" DO ...	the character Q
ON SYM DO ...	any string, terminated by carriage-return or tab
ON SYM "XYZ" DO ...	the string XYZ
ON NUM DO ...	any signed decimal integer
ON SW DO ...	any switch on the mouse
ON SW 2 DO ...	switch 2 on the mouse
ON HIT 27 DO ...	pointing at part of display procedure call 27 on the screen and pressing any switch.
ON PFLAG 6 DO . .	program flag 6 set
ON PFLAG DO ...	any program flag set

The meaning of most of these is fairly obvious. Strings may be of up to five characters only--this is one of the restrictions on string-manipulations. The mouse switches create inputs only when they are pressed

down, and not when they are released.

The "ON HIT..." statement introduces a further feature in Dial graphical output, the ability to associate names with display procedure calls. These names may be any integer, and can be added to both calls and WINDOW statements, as follows:

```
RESIS AT CX, CY AS 17;
DPROC[K] AT 200, 500 SIZE 2*K AS K+3;
WINDOW RESIS AT X, Y SIZE S AS N;
```

These names are used only in pointing operations, where they serve both to single out part of the picture to be pointed at, and to report back which part was "hit". For example, suppose a procedure P is called with name 1, and itself calls three other display procedures, Q, R and S with names 2, 3 and 4. Then if the user points with the mouse cursor at any part of the resulting picture and presses a switch, on execution of an "ON HIT 1 DO..." statement, the name of the indicated part will be reported back by means of a reserved variable HITN. This name will have the value 2, 3 or 4 according to the part chosen. P may include calls without names, but these parts of the picture cannot generate a "hit". Only names occurring one level below the name in the ON HIT statement will be reported back.

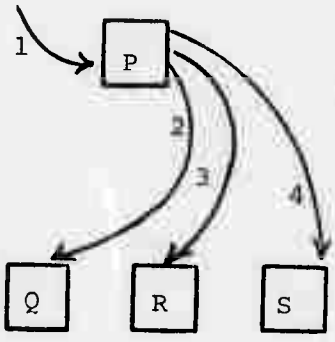


Figure 6

PROGRAM STATES

While a program is waiting for an input it is in an idle state, and as soon as an input occurs the program will normally execute a series of ON statements to determine the type of input and the appropriate action to take. The result may be that the program transfers to the head of a different list of ON statements to await further input. In this case the program is considered to have changed state.

The concept of program states has been built into Dial, in much the same fashion as in the PDP-9/PDP-10 Graphics System[2]. When the program starts executing it is implicitly in State 1. It can be made to change state by executing an ENTER statement, for example

```
ENTER 12;
```

The tasks that the program is to carry out during a given state are listed under a DURING statement:

```
DURING 12 DO
    BEGIN ON CHAR "A" DO ...
          ON CHAR "P" DO ...
          ON SW DO ...
    END;
```

ON statements should be used only within a DURING statement. Conversely, DURING statements should, in fact, contain a list of one or more ON statements; any other code included in the DURING statement but not within an ON may cause unexpected results.

Besides the DURING statement, there is an ENTERING statement for defining the operations to be carried out before transferring to the

DURING statement:

```
ENTERING 12 DO H ← 0;
ENTERING 5 DO BEGIN TYPE "PRESS SWITCH"; K ← K+1 END;
```

ENTERING and DURING statements may occur in any order within the program. Statements not enclosed within these ENTERING and DURING statements are executed when the program is started. Programs like the one on Pages 10 and 11 which contain no ENTERING or DURING statements simply run to completion and return to the monitor.

It is possible for the program to set one of the "program flags," and later to test for the flag with an "ON" statement. This creates a method for the program itself to cause branching from a state. For example:

```
......SET FLAG 6; ENTER 3;
.........
DURING 3 DO
ON PFLAG 6 DO BEGIN......
```

There are 262,144 possible flags including Flag 0. "SET FLAG" clears any other flag currently set. It is not necessary to include a flag number in the "ON PFLAG" statement:

```
ON PFLAG DO ....
```

This will branch if any flag is set. The flag number is stored in FLAGNO.

A statement "PAUSE" <STATEMENT> is included in Dial. This causes program FLAG Ø to be set N 60ths of a second later, where N is the value of the statement:

```
PAUSE 3Ø                    % WILL SET FLAG Ø AFTER 1/2 SEC %
```

The "PAUSE" statement does not halt computation. Instead it starts a time (in the PDP-9) which inputs a special character when the interval has elapsed. For example, the following statements will call procedure UPDATE and reconstruct frame F6 every 2 seconds:

```
SET FLAG Ø; ENTER 12;  % TO CAUSE 1ST BRANCHING %

DURING 12 DO
  ON PFLAG DO BEGIN PAUSE 12Ø;     % 2 SEC INTERVAL %
                    UPDATE;
                    FRAME F6
                  END;
```

Program Flag 0 can be set from the teletype by typing control + shift + M. "PAUSE" statements will work from a non-PDP-9 teletype, but intervals are rounded to the nearest second, and cause suspension of execution until the end of the interval.

RESERVED VARIABLES

The Dial system uses a number of reserved variables to pass data to the program during input operations. Four of these deal with input text information:

INCHAR	the latest character typed, shifted into the leftmost 7 bits
INSYM	the input symbol, excluding the terminating character (5 characters maximum)
INTEXT	the input string array (may be subscripted)
INVAL	the signed value of the number typed in

Whenever a switch is pressed on the mouse, its position and the switch settings are passed to the PDP-10, and are held in the following locations:

INX	mouse x-coordinate
INY	mouse y-coordinate
INSW	switch number (1, 2 or 3)

As mentioned above, when a hit is detected, the name of the display procedure call involved is held in HITN. Two other locations, HITX and HITY, contain the mouse position converted to the local coordinates of this display procedure: this helps to determine in which region of a display procedure the hit occurred.

Two reserved procedures, SCALX and SCALY, are included to help relate the mouse position to scaled pictures. Each has one argument, which is the name of a call to a display procedure. They return the

position of the mouse in the ccordinate system of this display procedure. For example:

```
...WINDOW CCT AT X1, Y1 SIZE 1000 AS 20;
.......
...X2 ← SCALX[20]; Y2 ← SCALY[20]; ...
```

will deposit in X2, Y2 the mouse position in the coordinate system of CCT, relative to the origin of CCT.

These routines assume that a procedure call exists in the display structure with the name given. If there is no such call, the values they return are indeterminate.

STRING MANIPULATION

Apart from strings included in output lists, the longest string that Dial can handle is five characters in length. Strings of up to this length can be treated like ordinary variables:

```
A[10] ← "JOHN"
IF Q3 = "DOG" THEN ...
```

Variables containing text information can be output in character form by preceding the variable name with a dollar sign:

```
Q ← "ABC" DISPLAY N, "WATCH THIS SPACE",$Q;
```

The TYPE statement for outputting to the teletype is identical in most respects to the DISPLAY statement. Both use the reserved string NEWLINE for carriage control. Displayed text is shown at a fixed size, which does not change when the scale of the picture changes. The left-hand margin for displayed text is the left-hand edge of the display procedure within which it occurs.

Since the compiler will not permit space, tab, carriage return or rubout to be included as a single character within quotes, four reserved variables are included which contain these characters as strings. These are called:

```
SPACE
TAB
NL              (Carriage return)
RUBOUT
```

E.g., `ON CHAR DO IF INCHAR=RUBOUT THEN A[K]←Ø;`

COMMENTS

Any text enclosed within percent signs is ignored by the compiler and may therefore be treated as comments. For example:

```
IF A > B THEN BEGIN          % DO THIS IF A GREATER THAN B %
    P[K] ← P[K+1];           % MOVE P VALUE DOWN %
    TYPE P[K+2]
    END;
```

LINKING TO OTHER PROGRAM SEGMENTS

Dial has been designed to link to other relocatable PDP-10 programs. An INTERNAL declaration must be used for any variable or array internal to the DIAL program which is addressed by another program segment. Because of the rather peculiar calling sequence for Dial procedures, it is difficult for MACRO-10 or FORTRAN programs to call them. On the other hand, DIAL programs can call MACRO-10 or FORTRAN subroutines:

```
CALL DSKOUT; CALL INVERT[A,B,X];
```

These generate the correct "JSA 16", calling sequence for FORTRAN. All the accumulators are saved before the call and restored afterwards, apart from accumulator 1 whose contents are treated as the value of the sub outine. This provides a means of writing external functions.

REFERENCES

1. Wirth, N. and Weber, H., "Euler: a generalization of Algol, and its formal definition". *Communications of the ACM*, Vol. 9 13-25+ and 89-100 (Jan. and Feb.).

2. Newman, W. M. "A high-level programming system for a remote time-shared graphics terminal". University of Utah, Computer Science, March, 1969.

3. Sutherland, I.E. "A head-mounted three dimensional display". AFIPS, *Proceedings of the Fall Joint Computer Conference*, Vol. 33, part 1, 1968, 757-764.

4. Newman, W. M. "Programming guide to the UNIVAC 1559 display". University of Utah, Information Research Laboratory, October, 1969.

5. Kilgour, A. C. and Brown, M. D. "SPINDLE: a system permitting interactive display list editing". University of Edinburgh, Computer-aided design Project, June, 1969.

APPENDIX I: A DETAILED DESCRIPTION OF THE DIAL LANGUAGE

TITLE

The first line of a Dial program must be a title:

```
TITLE PROG 1
BEGIN ....
```

DECLARATIONS

There are three types of declaration: NEW, DISPLAY PROCEDURE, and INTERNAL. These should be used as follows:

NEW	for all variables, arrays, procedures and frame procedures which occur in the program;
DISPLAY PROCEDURE	for all display procedures;
INTERNAL	for all variables and arrays in the program which are referenced by other programs. These should also be declared in a NEW declaration.

Declarations must be placed at the start of the program immediately following the initial BEGIN. Any number of declarations may be be included, in any order.

Arrays may have any number of dimensions. All such dimensions have a lower bound of 1. Dial arrays are compatible with FORTRAN arrays.

Display procedure declarations may include size definitions, denoted by an integer followed by a colon. All procedures whose name follows such a definition are given that size, up to the next size definition. Those not preceded by a size are given a size of 1024_{10}.

Examples:

```
TITLE PROG 2
BEGIN NEW A,B[3,100], MAX, MIN, FRAME1;
     INTERNAL A,B;
     DISPLAY PROCEDURE PLAN1,32: DOOR, WINDOW;
```

PROCEDURE DEFINITIONS

Procedure definitions may be placed anywhere within the program, but must precede any call to that procedure. The body of the procedure definition, including a list of its formal parameters if any, is enclosed within single quotes:

```
MAX ← 'NEW ARG1, ARG2;
    IF ARG1 > ARG2 THEN ARG1 ELSE ARG2'
```

When a procedure is called, each of the arguments is in turn evaluated, and its value is assigned to one of the formal variables in the NEW list, starting with the first. There may be more formal parameters in the list than values passed, in which case the remaining formals are not given values, and can in fact be used as local variables by the procedure. The value returned by a procedure is the value of the last executed statement. For example, the following procedure would return the value 3:

```
TYPEB ← 'TYPE B; X ← 3'
```

EXPRESSIONS

Arithmetic expressions may contain any number of primes separated by the symbols + - * / signifying addition, subtraction, multiplication and division. When the expression is evaluated, multiplication and division are carried out first.

The following are legal primes:

1. Integer constants
2. Variables
3. Single array elements
4. Procedure calls
5. Any statement, enclosed within parentheses

Any alternative form of expression is a string of one to five characters, enclosed within double quotes. This has the value of the integer containing these characters in 7-bit ASCII format, left justified.

Examples of permissible expressions are:

```
123
A6
-A+B-3
(A+B-3)/(16-C)
C+(IF A=0 THEN 12 ELSE 13)
H+J[16]
MAX[B,C]
"JOHN"
```

In the above, any of the identifiers might refer to procedures rather than to variables or arrays. Any statement can be used as the argument of a procedure or array call.

ASSIGNMENT STATEMENTS

An expression or statement, preceded by a left arrow pointing to the name of a variable or of an array element, constitutes an assignment statement:

```
A ← B + C
TXT[3] ← "HENRY"
```

These statements have the value obtained by evaluating the right-hand side. This value can be multiply assigned:

A ← B ← XYZ[21] ← Ø

CONDITIONAL STATEMENTS

Three forms of conditional statements are permitted:

IF <logical expression> THEN <statement>
IF <logical expression> THEN <statement> ELSE <statement>
WHILE <logical expression> DO <statement>

The basic logical expressions are the following:

E1=E2	true if value of E1 equals value of E2
E1#E2	true if value of E1 not equal to value of E2
E1>E2	true if value of E1 exceeds value of E2
E1<E2	true if value of E1 is less than value of E2
E1>=E2	true if value of E1 exceeds or equals value of E2
E1<=E2	true if value of E1 is less than or equals value of E2

More complex logical expressions can be created by means of the operators OR, AND and NOT.

e.g., IF A>C AND NOT C=D THEN ....

NOT is applied first during evaluation, then AND, finally OR.

If the logical expression in an IF statement is found to be true, the statement following the THEN is executed; otherwise the ELSE statement, if any, is executed. In a WHILE statement, the statement following the DO is executed repeatedly as long as the logical expression is true.

Examples: IF A>B THEN P←0 ELSE Q←1
IF A[12] # A[13] THEN A[12]←A[13]←0
WHILE K<0 DO K←K + 1

The expression INCHAR=NL is true if INCHAR contains the single character, carriage-return.

FOR STATEMENT

The following are examples of permissible FOR statements:

```
FOR K ← 1 STEP 1 UNTIL 100 DO A[K] ← 0
FOR P ← J+3 STEP B + 100*Q UNTIL 7-J/4 DO A[P] ← N[P-1]
```

Any expression may precede the words STEP, UNTIL and DO; any statement may follow DO. This statement will be executed while the variable proceeds from a value equal to the first expression to a value not exceeding the last expression, in steps equal to the middle expression.

BLOCKS

A compound statement can be formed from any number of statements, separated by semi-colons, preceded by BEGIN and terminated by END.

```
e.g.,   A ← BEGIN TYPE "STRING"; C ← 0; 3 END
```

INPUT STATEMENTS

All input to a Dial program must occur within a DURING statement. This has the form:

```
DURING <state number>  DO  <statement>
```

and is equivalent to a WHILE statement:

```
WHILE STATE = <state number>  DO
    BEGIN INPUT; <statement>  END
```

While the program is under the control of the DURING statement it is waiting for input from the Teletype or from the Mouse. When it receives one of these it executes the following statements. These are normally ON statements:

```
ON  <input>  DO  <statement>
```

for which alternatives are given on Page 12 of the main report.

ON statements are executed in the order in which they are given, until one is found which matches the input. For this reason it is important not to program as follows:

```
ON SYM DO ENTER 3;
ON SYM "ABC" DO A ← 3;
```

The second of these statements can never be satisfied, since the first will always be satisfied before it.

To cause the program to transfer to another state, the ENTER statement is used:

```
ENTER 23
```

The argument of this statement may itself be any statement:

```
ENTER P+3
```

ENTER statements take immediate effect, and act rather like GO TO statements. However, the action is not identical to a GO TO: instead the current state number is reset, and all the DURING statements in the program are executed until the appropriate one is found. There may be no DURING statement bearing the desired state number. In this case the program returns to the monitor.

It may be useful to check for certain inputs during all states. For this purpose a special DURING statement is provided:

```
DURING ALL DO ON CHAR "X" DO ENTER 1
```

The check for inputs listed in the DURING ALL statement is made before checking any inputs listed under the current state.

ENTERING statements can be used to cause tasks to be performed every time a certain state is entered:

```
ENTERING 3 DO A ← B ← Ø
```

ENTERING and DURING statements may be listed in any order in the body of the program. When execution starts, all the statements not enclosed in ENTERING and DURING statements are executed, and the program enters State 1. This last action can be avoided by including an ENTER statement in this initialization section.

OUTPUT STATEMENTS

For text output, the TYPE statement is used. Following the word TYPE is a list of expressions, which are evaluated and typed out. Strings in a type list may be of any length, but may not include carriage-returns: these are provided by the reserved NEWLINE string:

```
TYPE A, "SQUARE FEET", NEWLINE;
```

Variables or array elements whose names are preceded by a dollar sign are typed out as ASCII strings:

```
XT ← "ED"; TYPE $XT
```

The DISPLAY statement has the equivalent effect to the TYPE statement on the display screen. All characters are displayed at a standard size, roughly the same as the size of typed characters. The following two forms are permitted for the DISPLAY statement:

```
DISPLAY "TEXT", N
DISPLAY "TEXT", N AT X,Y
```

Other statements for generating displayed pictures are:

```
LINE Δx, Δy
MOVE Δx, Δy      } by relative amounts
ZIP  Δx, Δy
```

LINE TO x,y
MOVE TO x,y
ZIP TO x,y

} to absolute positions

Any statement may be used for $\Delta x, \Delta y$, x and y. These value are measured in units of the coordinate system of the display procedure in which the statements occur. ZIP lines are drawn entirely in zip mode.

There are four forms of display procedure call:

```
DP AT X,Y
DP AT X,Y SIZE S
DP AT X,Y AS N
DP AT X,Y SIZE S AS N
```

Here X, Y, S and N are any statement. The call DP may include arguments.

The WINDOW statement takes similar forms:

```
WINDOW DP AT X,Y SIZE S
WINDOW DP AT X,Y SIZE S AS N
```

The center of the window is given by X,Y. The window is always square with side equal to 2S. Those parts of DP which lie within the window are mapped across into the visible screen area.

Any procedure containing any number of WINDOW statements can be called as a FRAME procedure:

```
FRAME FR1
```

It is wise not to include arguments in frame procedure calls, altho gh these will currently be processed correctly; no guarantee can be made that this will continue to be the case. Each time a frame procedure is called in this way, a fresh display file replaces the previous output of this frame. The statement

```
DELETE FR1
```

will delete this frame's output, and

```
DELETE ALL
```

will clear the screen.

CALLING EXTERNAL SUBROUTINES

The following statements can be used to call external subroutines written in FORTRAN IV or MACRO-10.

```
CALL WTAPE
CALL PARAB[100, A + 100 * K]
```

Arguments may be variable or array names, or any other statement, including another CALL statement. Accumulators are saved before the call, and restored afterwards. All subroutines called in this way must be declared in EXTERNAL declarations.

COMMENTS

Any text enclosed within percent signs is ignored by the compiler.

APPENDIX II: USE OF THE DIAL SYSTEM

Source files for Dial programs should be prepared in the normal fashion, using one of the editors. The procedure for compiling and running Dial programs is as follows:

```
.R DIAL
*DSK:PROG.MAC←DSK:PROG.SRC
                (any file names and any devices,
                 including TTY, will do)
.LOAD PROG,DIO[1,1]
                (this loads the program, together with
                 the Dial I/O routines)
.START
```

The program will now start execution.

Dial programs can be run from any teletype, but will not produce any displayed output unless run from one of the teletypes on the PDP-9. To remind users who use other teletypes, the program types "TTY IO ONLY" when it starts execution.

To help programmers to debug without a display, a technique for simulating the mouse from the keyboard is included in Dial. Type control + shift + N, followed by up to three integers, separated by commas. The first number is taken as the x-coordinate of the mouse, the second as the y-coordinate and the third as the switch number. For example:

↑C256,891,2 is equivalent to moving the mouse to the point 256,891 and pressing the middle button.

Because of the way Dial processes hits, these can be generated at the teletype in the same way.

Those who wish to examine the display files produced by Dial should do the following:

```
.DEBUG PROG, DIO[1,1]
DIO$:  BP(DBUF)$1B    $G
```

On reaching the breakpoint set in this way, DDT will type out the first word of the display file it has prepared for transmission, and ensuing words can be examined by typing line-feed. To resume execution, type $P ($ = altmode in all cases). Those who use this technique should familiarize themselves with the instruction set for the Univac 1559 [4].

APPENDIX III: FILE I/O IN DIAL

I. Input

A. Initialization (Opening)

There are two calls to open an input device. Only one input device can be open at any time. Each open input call closes the previous input device, if one was open.

1. TXTIN [Open device for character input]

1A. Parameters

This call takes three text string parameters. The first is the device name, the second is the file name, and the third is the file name extension, e.g., CALL TXTIN["D116","SCAN","MAC"];

1B. Results

This call opens the device named in the first parameter, and looks up the file name specified in the next two parameters.

Code is also set up to access the 36-bit words from this device in 7-bit character bytes.

2. BININ [Open device for binary input]

2A. Parameters

This call takes three text string parameters. The first is the device name, the second is the file name, and the third is the file name extension, e.g., CALL BININ ["D116","SCAN","MAC"];

2B. Results

This call opens the device named in the first parameter, and looks up the file name specified

in the next two parameters

Code is also set up to access the 36-bit words from this device in 36-bit binary bytes.

B. Reading

There are two calls provided to read an input device, and one to check if end of file has been reached.

1. RDTXT [Read characters]

1A. Parameters

This call takes no parameters. It simply leaves a value on the top of the stack.

E.g., CHARS ← CALL RDTXT;

1B. Results

The resulting word will be the next five non-zero bytes from the input device. If the device was opened to have 36 bit bytes, then only the seven low-order bits will be taken from each non-zero byte.

2. RDBIN [Read binary words]

2A. Parameters

This call takes no parameters. It simply leaves a value on the top of the stack.

E.g., WORD ← CALL RDBIN;

2B. Results

The resulting word will be the next byte from the input device. If the device was opened to have seven bit bytes, then the byte will be right justified in result word.

3. EOF [Check for end of file]

3A. Parameters

This call takes no parameters. It simply leaves a value on the top of the stack.

E.g., EOFVAL ← CALL EOF;

3B. Results

The value is zero if end of file has not been reached. The value is set to -1 when end of file is reached. Note that the binary word returned when EOF is set is not part of the file, and always zero. Also the text string is part of the file, but may be filled out with nulls [Ø].

C. Termination (Closing)

There is one call to close the current input device which takes no parameters. It is CLOSEI.

II. Output

A. Initialization (Opening)

There are two call to open an output device. Only one output device can be open at any time. Each open output call closes the previous output device, if one was open.

1. TXTOUT [Open device for character output]

1A. Parameters

This call takes three text string parameters. The first is the device name, the second is the file name, and the third is the file name extension.

E.g., Call TXTOUT ["DI16","SCAN","MAC"];

1B. Results

This call opens the device named in the first parameter, and enters the file name specified in the next two parameters. Code is also set up to

store the 36-bit words sent to this device as 7-bit character bytes.

2. BINOUT [Open device for binary output]

2A. Parameters

This call takes three text string parameters. The first is the device name, the second is the file name, and the third is the file name extension.

E.g., CALL BINOUT ["D116","SCAN","MAC"];

2B. Results

This call opens the device named in the first parameter, and enters the file name specified in the next two parameters. Code is also set up to store the 36-bit words sent to this device as 36-bit binary bytes.

B. Writing

There are two calls to write on an output device.

1. WRTTXT [Write characters]

1A. Parameters

This call takes one parameter, which is the data to be written. E.g., CALL WRTTXT[DATA];

1B. Results

This will cause the data word to be divided into 5 7-bit characters, and each non-zero character will be written. Note, that if the output device was opened in binary mode, each character will be stored right-justified one to a word.

2. WRTBIN [Write binary words]

2A. Parameters

This call takes one parameter, which is the data to be written. E.g., CALL WRTBIN[DATA];

2B. Results

This will cause the entire data word to be written at once. Note, that if the output device was opened in character mode, only the seven low order bits will be written in the next 7-bit slot in the output file.

C. Termination (Closing)

There is one call to close the current output device, which takes no parameters. It is CLOSEØ.

III. Use

Programs should be loaded with DIAL10 as follows:

.LOAD PROG, DI0[1,1], DIAL10[1,1]

APPENDIX IV: AN EXAMPLE OF A DIAL PROGRAM

The following program allows the user to create and manipulate rectangles on the screen and illustrates most of the interactive features of Dial. The functions of the program have been based on an example in A. C. Kilgour and M. D. Brown's SPINDLE Manual.

Rectangles can be created, moved, duplicated, and deleted. To create a rectangle, type "G" and indicate two positions with the mouse. These become the two opposite corners of the rectangle. Each further point indicated by the mouse will be used to reposition the rectangle.

To move a rectangle, type "M" and point at the rectangle with the mouse: the rectangle will disappear. Then point somewhere on the screen, and it will reappear at that position. At each successive position, the rectangle is repositioned.

Duplication is achieved by typing "S" and pointing at the appropriate rectangle. Thereafter the operation is identical to moving, except that the original rectangle remains on the screen.

To delete a rectangle, type "D" and point at it. If deleted in error, a rectangle can be restored by typing "N", and a fresh choice can be made.

Throughout operation, typing "X" will return the program to its base mode, and restore the display to the state it was in when base mode was left. Typing "P" makes permanent any change effected by creating moving, duplicating or deleting, and returns the program to base mode.

The program starts with an implicit "G".

```
TITLE RECTS
BEGIN
NEW NR,K,TEMP,NEWX,NEWY,SIZX,SIZY,CREATE;
NEW RSIZ[5Ø,2],RX[5Ø],RY[5Ø],FR1,FR2;
DISPLAY PROCEDURE NEWR,RECT,LAYOUT;

% RECT IS CALLED BY LAYOUT TO DRAW ALL STORED RECTANGLES %

RECT←'NEW N; LINE  RSIZ[N,1],Ø; LINE Ø, RSIZ[N,2];
            LINE -RSIZ[N,1],Ø; LINE Ø,-RSIZ[N,2]';

% LAYOUT CREATES DISPLAY OF ALL STORED RECTANGLES, OMITTING
  TEMPORARILY DELETED RECTANGLE TEMP %

LAYOUT←'FOR K←1 STEP 1 UNTIL NR DO
       IF K#TEMP THEN RECT[K] AT RX[K],RY[K] AS K';

% NEWR DRAWS NEWLY DEFINED RECTANGLE %

NEWR←'MOVE TO NEWX,NEWY;
     LINE SIZX,Ø; LINE Ø,SIZY; LINE -SIZX,Ø; LINE Ø,-SIZY';

% FRAME FR1 CREATES ALL STORED RECTANGLES,
  FR2 CREATES NEW RECTANGLE %

FR1←'WINDOW LAYOUT AT 512,512 SIZE 512 AS 51';
FR2←'WINDOW NEWR AT 512,512 SIZE 512';

% THIS CODE EXECUTED AT START %

CREATE←TEMP←NR←Ø
ENTER 2;

% BASE MODE STATE %

DURING 1 DO
BEGIN ON CHAR "G" DO ENTER 2;
      ON CHAR "M" DO ENTER 5;
      ON CHAR "S" DO ENTER 6;
      ON CHAR "D" DO ENTER 7
END;

DURING 2 DO
  ON SW DO BEGIN NEWX←INX;              % FIRST CORNER %
                 NEWY←INY;
                 ENTER 3
           END;

DURING 3 DO
  ON SW DO BEGIN SIZX INX-NEWX;         % SECOND CORNER %
                 SIZY INY-NEWY;
                 FRAME FR2;             % DISPLAY NEW RECTANGLE %
                CREATE 1;               % IE NEW RECT CREATED %
                ENTER 4
```

```
            END;

DURING 4 DO
  ON SW DO BEGIN NEWX←INX;          % ON EACH NEW POSN, REPOSITION %
                 NEWY←INY;
                  FRAME FR2         % AND DISPLAY IN NEW POSITION %
            END;

DURING 5 DO
  ON HIT 51 DO BEGIN TEMP←HITN;     % TEMP = RECTANGLE TO BE MOVED %
                     SIZX←RSIZ[HITN,1];
                     SIZY←RSIZ[HITN,2];
                     FRAME FR1;     % DISPLAY WITHOUT THIS RECTANGLE %
                     CREATE←1;
                     ENTER 4
                END;

DURING 6 DO
  ON HIT 51 DO BEGIN SIZX←RSIZ[HITN,1];  % DUPLICATE THIS ONE %
                     SIZY←RISZ[HITN,2];
                     CREATE ←1;
                     ENTER 4
                END;

DURING 7 DO
  ON HIT 51 DO BEGIN TEMP←HITN;          % DELETE THIS ONE %
                     FRAME FR1;
                     ENTER 8
                END;

DURING 8 DO
  ON CHAR "N" DO BEGIN TEMP←Ø;           % N MEANS RESTORE IT %
                       FRAME FR1;
                       ENTER 7
                  END;

DURING ALL DO
  BEGIN ON CHAR "P" DO              % P MEANS MAKE CHANGE PERMANENT %
    BEGIN IF TEMP#Ø THEN
      BEGIN NR←NR-1;                % TEMP = RECTANGLE TO BE DELETED %
        FOR K←TEMP STEP 1 UNTIL NR DO
        BEGIN RSIZ[K,1]←RSIZ[K+1,1];     % FILL GAP LEFT BY DELETING %
              RSIZ[K,2]←RSIZ[K+1,2];
              RX[K]←RX[K+1];
              RY[K]←RY[K+1]
        END;
        TEMP←Ø
      END;
      IF CREATE=1 THEN
      BEGIN CREATE←Ø;               % IF NEW RECTANGLE CREATED, ADD IT %
        NR←NR+1;                    % AND INCREMENT NUMBER %
        RSIZ[NR,1]←SIZX;
        RSIZ[NR,2]←SIZY;
        RX[NR]←NEWX;
```

```
        RY[NR]←NEWY
      END;
      DELETE FR2;                          REMOVE TEMPORARY DISPLAY %
      FRAME FR1;
      ENTER 1
    END;
    ON CHAR "X" DO                         % ON X, RESTORE TO ORIGINAL STATE %
    BEGIN DELETE FR2;
          FRAME FR1;
          CREATE←TEMP←Ø;
          ENTER 1
    END
  END
END
```