Contains † & ., with structure

Chapter 5 1

STATEMENTS 1a

5.1 Concept of Statements 1a1

†Statements are the operational units of JOVIAL. They describe self-contained rules of computation specifying manipulations of data, and they describe the sequencing, conditional or unconditional, of the execution of †statements. 1a1a

.1 Wherever the syntax says that a †statement can be used, any of the above kinds of †statement can be used. The term †controlled:statement is sometimes used in describing other †statements. A †controlled:statement is not any special kind of †statement but is a required part of a †conditional:statement or †loop:statement. 1a1a1

.2 Any †statement may be used where a †controlled:statement is specified--except for the particular forms prohibited in the description of the †conditional:statement. 1a1a2

.3 The kinds of †simple:statements listed below are discussed in Sections 5.5 through 5.14: 1a1a3

.4 Some †simple:statements such as the †loop:statement, †switch:statement, and †conditional:statement may contain †compound:statements. 1a1a4

5.2 †Null:Statement 1a2

In some language forms defined in this manual, sme preliminary structure is followed by a single †statement. A †simple:statement or †compound:statement usually completes the form. If there is no significant †statement desired in such a case, a †null:statement can be used to complete the form. 1a2a

.1 The second form given above for †null:statement is a single †null:statement regardless of whether the optional †semicolon is included. A single optional †semicolon is permitted as a terminator for a †direct:statement, a †switch:statement, †compound:declaration, or a †compound:statement. If used, this †semicolon merely terminates the †statement or †declaration and is not a †null:statement. Another use for the †null:statement is given in the discussion of †switch:statement. 1a2a1

5.3 †Compound:Statement 1a3

A †compound:statement is a †statement whose essential character is bound up in containing one or more other †statements as a part of itself. A †compound:statement (like a †loop:, †conditional:, or †switch:statement) is a grouping of other †statements. 1a3a

.1 In the syntax equation above, there must be at least one †statement other than a †null:statement between ←BEGIN and ←END for it to be considered a †compound:statement. †Directives may be included in a †compound:statement (see Chapter 11). The optional †semicolon, if given, does not constitute a †null:statement; it merely terminates the †compound:statement. An example of a †compound:statement that contains a †compound:statement is: 1a3a1

```
←BEGIN      IF J > 9 ;                          1a3a1a
←  BEGIN    J = J + 1 ;    GOTO OUT ;           1a3a1b
←  END                                          1a3a1c
←END                                            1a3a1d
```

5.4 †Named:Statement 1a4

Any †statement can be named. 1a4a

.1 A †statement:name is defined by attaching the †name followed by a †colon to any †statement. The †statement to which the †name is attached becomes thereby a †named:statement. The †named:statement retains its character as a †null:statement, †simple:statement, or †compound:statement when the †name is attached, and so another †name can be attached. Example: 1a4a1

```
←CEASE:  DESIST:  HALT:  STOP;                  1a4a1a
```

.2 The above example is a †stop:statement that has three †names, ←CEASE, ←DESIST, and ←HALT. The †stop:statement is also a †named:statement and a †simple:statement. More examples of naming various kinds of †statements follow: 1a4a2

```
←EPSILON:  ZETA = ETA;                          1a4a2a
←THETA:    IF   . . . (uncompleted)             1a4a2b
```

```
←IOTA:      FOR  . . . (uncompleted                          1a4a2c

←KAPPA:     BEGIN . . . (uncompleted)                        1a4a2d

←LAMBDA:    TEST;                                            1a4a2e

←MU:        EXIT KAPPA;                                      1a4a2f
```

.3 A †statement:name may be attached to two structures that are not †statements. Specifically, a †statement:name may precede ←ELSE, and a †statement:name may precede the first ←BEGIN following ←SWITCH, as in these examples: 1a4a3

```
←NU:        ELSE    ...   (uncompleted)                      1a4a3a

←SWITCH XI-3; ←MICRON:   BEGIN  ... (uncompleted)            1a4a3b
```

.4 The effect of references to ←NU is exactly the same as if ←NU were attached to the †statement following ←ELSE. The effect of a †go:to:statement referencing ←MICRON is explained in Section 5.12.1. The effect of an †exit:statement referencing ←MICRON is the same as if ←MICRON were attached to the †switch:statement. 1a4a4

5.5 †Assignment:Statements, †Exchange:Statement 1a5

A †simple:assignment:statement specifies that the †formula to the right of the †equals:sign be evaluated and that this value become the new value of the †variable to the left of the †equals:sign. It is permissible for the †variable on the left to occur also in the †formula on the right. In this case, the old value of the †variable is used in the calculations needed to evaluate the †formula. The †formula is evaluated as described in Chapter 4 (see particularly Sections 4.15 and 4.16), then any †index or †pointer:formula associated with the †variable on the left is evaluated, and the value of the †formula is assigned to the †variable. Any reordering of the computations for optimization is prevented if an †order:directive precedes the †assignment:statement or the †declaration of a function for which a †function:call occurs in the †formula. In the forms: 1a5a

```
←  BIT      ←(†formula←, †numeric:formula    ←,
†numeric:formula   ←)                                        1a5a1

←  BYTE                                                      1a5a2
```

the leftmost †formula, including any †indices or pointers,

is evaluated first, then the second and then the third, if it exists, to determine the parts of the first †formula to use. 1a5b

.1 Assignment of a †formula of any type is permitted to a †variable of any type. Conversions and scaling are performed as needed when the operands are of different numeric types. If the types seem incompatible, the †formula on the right becomes a †bit:formula; then the bits of this †bit:formula replace the bits of the †variable on the left. If the †bit:formula has more bits than the †variable to which it is being assigned, leading bits are truncated; if there are too few bits, leading zeros are supplied before assignment. In assigning to the †functional:variable beginning with ←BIT, the bits of the †formula on the right, whatever its type, are assigned to the specified bits of the †variable, right justified. 1a5b1

.2 After a value has been obtained from the †formula on the right, any †index and †pointer:formula for the †variable to be assigned are evaluated. If the form on the left is: 1a5b2

←BIT ←(†named:variable←, †numeric:formula ←,†numeric:formula ←) 1a5b2a

←BYTE 1a5b2b

any †index and †pointer:formula the †named:variable bears are evaluated first, then the second and third (if any) formulas, before assignment takes place. 1a5b3

.3 When assigning a †character:formula to any †variable not a †character:variable, it first becomes a †bit:formula and is assigned as a bit string, right justified, and truncated on the left or padded on the left with zeros if necessary. In assigning a †character:formula to a †character:variable, if the †formula is too long, excess bytes on the right are truncated before the assignment. If the †formula is too short, blank characters are added at the right to match the size of the †variable before the assignment. 1a5b4

.4 The more general †assignment:statement permits multiple †variables to be assigned. These †variables may be individually listed or a sequence of occurrences of a †variable may be indicated by using an †indexed:variable:range (see Section 10.4.4), or some

combination of these †variable forms may be used. The forms of the †assignment:statement that use †format:variable and †format:function:call are discussed in Chapter 6 on formatting. Omitting the forms related to formatting gives an †assignment:statement 1a5b5

```
   variable                    ←     ←=     formula
←    ←;                                                    1a5b5a

   indexed:variable:range
indexed:variable:range                                     1a5b5b
```

.5 When the †indexed:variable:ranges are expanded to the sequences they represent, the above form is equivalent to: 1a5b6

```
variable     ←      ←=   formula     ←      ←;              1a5b6a
```

which closely resembles the †simple:assignment:statement. In this †assignment:statement all of the †formulas are evaluated before any assignments are made. Then the value of the leftmost †formula is assigned to the leftmost †variable and the value of each †formula is assigned to its corresponding †variable--corresponding by position in the list. There must be at least as many †variables to the left of the †assignment:operator as there are †formulas to the right. If there are fewer †formulas than †variables, the value of the rightmost †formula is assigned to all the extra †variables at the right of the list of †variables. The leftmost †formula is evaluated first, then the next †formula, and so forth. After all †formulas are evaluated, any †index or †pointer:formula for the leftmost †variable is evaluated and the †variable is assigned, then any †index or †pointer:formula for the next †variable is evaluated and that †variable is assigned a value, and so forth. 1a5b7

.6 The handling of ←BIT or ←BYTE, conversions, and type considerations are the same as for a †simple:assignment:statement. For a †formula to be assigned to several †variables, these considerations apply independently to each assignment. The following example illustrates an †assignment:statement 1a5b8

```
←AA[ 8:10], BB = AA[ 2:4 ];                                1a5b8a
```

.7 This is equivalent to four †simple:assignment:statements: 1a5b9

```
    .AA[8] = AA[2];    AA[9] = AA[3];                          1a5b9a

    .AA[10] = AA[4];    BB = AA[4];                            1a5b9b
```

.8 The †assignment:statement below uses a special form for an †indexed:variable:range 1a5b10

```
    .ALL(TAB) = 0;                                             1a5b10a
```

.9 This †statement causes every entry of table .TAB to be cleared to zero. It should not be used unless the programmer knows that the entire table is core resident at this time even though it may be allocated in increments. If some occurrences of the †pointer:variable point to the same submanifold it wastes time. If they point to other data, some unwanted clearing may occur. 1a5b11

.10 The †exchange:statement specifies that the old value of each of the two †variables is to become the new value of the other †variable. Any index or pointer for the †variable on the left is evaluated, then any †index or pointer for the †variable on the right is evaluated, and finally the values of the †variables are interchanged. The remarks made for †simple:assignment:statement concerning conversion, †variable type, and handling of .BIT or .BYTE apply also to the †exchange:statement. The following example of an †exchange:statement: 1a5b12

```
    .AA[I] == AA[I+J];                                         1a5b12a
```

is equivalent to the three †simple:assignment:statements 1a5b13

```
    .TEMP = AA[I];                                             1a5b13a

    .AA[I] = AA[I+J];                                          1a5b13b

    .AA[I+J] = TEMP;                                           1a5b13c
```

5.6 †Zap:Statement 1a6

Execution of a †zap:statement causes all items of all entries of the designated table, or (in the second form) all items of the designated †entry:variable to be set to null values. Character items are set to blanks; numeric items are set to zeros of the appropriate type. If items are overlaid (or share bits because of †specified:table:item:declarations) and such sharing results in conflicting values for some bits, the effect of .ZAP on such bits is undefined. If there are bits in an entry not

affected by any †item:description, the effect of ←ZAP on such bits is undefined. Like ←ALL, ←ZAP must never be used on a †table:name unless the entire table is core resident at the time even though it may be allocated in increments. 1a6a

5.7 †Conditional:Statement 1a7

The †conditional:statement provides for the conditional operation of a †statement or †statements based on the value of a †conditional:formula. 1a7a

.1 In either position in the definition of †conditional:statement, †controlled:statement is any one †statement, including a †conditional:statement, †switch:statement, †compound:statement, or †loop:statement. ←BEGIN and ←END brackets are not required unless it is desired that a group of two or more †statements constitute a †controlled:statement (or in the situation described in Section 5.7.8). The optional †statement:names preceding ←ELSE have the same effect as if they followed ←ELSE. 1a7a1

.2 The value of a †conditional:formula is the value of its rightmost bit after all operations called for in the †formula have been performed. The effect of a †conditional:statement depends upon whether there are one or two †controlled:statements. If there is but one, that †statement is executed if the value of the †conditional:formula is ←1 and is otherwise skipped. If there are two †controlled:statements, the first one is skipped when the †conditional:formula has the value ←0 and the second one is skipped when the value is ←1. Whenever there are two †controlled:statements in a †conditional:statement, only one is executed. Following the execution of the appropriate †controlled:statement, program execution normally continues from the point immediately following the †conditional:statement. There are exceptions discussed in Section 5.7.7 and shown in the examples of Section 5.7.8. 1a7a2

.3 Following are two examples of †conditional:statements 1a7a3

```
a.  ←IF ALPHA - BETA < 2 ;                                  1a7a3a
←           GOTO NEAR       ;                               1a7a3b
b.  ←IF BOOL ;                                              1a7a3c
←LBL:  BEGIN                                                1a7a3d
```

```
        ←RANDOM(:EASIC);                                    1a7a3e
        ←BASIC = EASIC**2;                                  1a7a3f
        ←END                                                1a7a3g
    ← ELSE                                                  1a7a3h
    ←L2:  BASIC = 0 ;                                       1a7a3i
```

.4 A list of alternatives may be encoded by nesting †conditional:statements. This can best be shown by example: 1a7a4

```
    ←IF ALPHA < BETA ;                                      1a7a4a
    ←   ALPHA = BETA ;                                      1a7a4b
    ←ELSE                                                   1a7a4c
    ←L1:   IF ALPHA + BETA > 10 ;                           1a7a4d
    ←              BEGIN                                    1a7a4e
    ←              GAMMA = (ALPHA + BETA)/2 ;               1a7a4f
    ←L2:           ALPHA = GAMMA+1 ;                        1a7a4g
    ←              BETA = GAMMA-1 ;                         1a7a4h
    ←              END                                      1a7a4i
    ←     ELSE GOTO KEEP ;                                  1a7a4j
    ←L3:                                                    1a7a4k
```

.5 The above example provides for the execution of one †assignment:statement if the first †conditional:formula is satisfied. It makes no difference then if the other †conditional:formula is satisfied. After execution of the single †assignment:statement the execution sequence continues with the statement at ←L3. If the first †conditional:formula is not satisfied, the second †conditional:formula is examined, (and so forth). A jump to ←L1 from elsewhere in the program causes the search for alternatives to begin at the point; it is as if execution of the †conditional:statement began at the top, but that all the †conditional:formulas before the referenced †name were false. A jump to ←L2 causes

execution of the †statement at the point regardless of the satisfaction of the earlier †conditional:formulas. In this case only two of the three †simple:statements that constitute the †controlled:statement are executed. Following execution of ←BETA = GAMMA - 1 ; control passes to the statement at ←L3. 1a7a5

.6 In constructing a nested †conditional:statement it must be remembered that each ←ELSE is matched with the nearest preceding unmatched ←IF at the same level of ←BEGIN END nesting. For example, in 1a7a6

```
←IF ...; ...; IF ...; ...; ELSE ...;
```
1a7a6a

the ←ELSE matches with the second ←IF, and in 1a7a7

```
←IF ...; BEGIN IF ...; ...; END ELSE ...;
```
1a7a7a

the ←ELSE matches the first ←IF because the second ←IF is not at the same level of ←BEGIN END nesting as the ←ELSE. 1a7a8

.7 In analyzing the flow of control through a complicated †conditional:statement, start with the innermost †conditional:statement matching ←ELSE's with ←IF's. Remember that an ←ELSE is always immediately preceded and immediately followed by a †controlled:statement, one and only one of which will be executed depending upon the value of the †conditional:formula of the †statement to which the ←ELSE belongs. Jumps must be generated around the †controlled:statement following a †conditional:formula if the value of the †formula is zero and around the †controlled:statement following any ←ELSE if the †controlled:statement preceding the ←ELSE is executed even in part. This may require a series of jumps and it is hoped that a compiler will optimize a chain of unconditional jumps by using the final destination with the original jump. 1a7a9

.8 Some nestings of †conditional:statements are not permitted. An embedded †conditional:statement without ←ELSE cannot be the †controlled:statement preceding ←ELSE within an encompassing †conditional:statement since this would permit an ←ELSE to match with an ←IF for which it is not intended. In nested †conditional:statements, a †conditional:statement without ←ELSE cannot be the †controlled:statement between two occurrences of the word ←ELSE. Of course, the effect can be achieved by appending ←ELSE NULL; (or ←ELSE ;) or by enclosing the

short statement in ←BEGIN and ←END brackets. The following two †conditional:statements are legal and have the same effect: 1a7a10

```
←IF A; IF B; IF C; E=1; ELSE ; ELSE IF D; E=2; ELSE ;
ELSE E=3;                                                    1a7a10a

←                    C1      C2            D1       D2     1a7a10b

                       ←B1                        ←B2       1a7a10c

                                    A1             A2       1a7a10d
```

```
←IF A; IF B; BEGIN IF C; E=1; END ELSE BEGIN IF D;
E=2; END ELSE E=3;                                           1a7a10e

                                ←C1
←D1                                                          1a7a10f

      ←B1                                        ←B2         1a7a10g

                                      ←A1
←A2                                                          1a7a10h
```

.9 In the examples above ←A1 and ←A2 show the first and second †controlled:statements of the †conditional:statement starting ←IF A, ←B1 and ←B2 show the first and second †controlled:statements associated with ←IF B, etc. The flow diagram below applies to either of the above †conditional:statements and shows the required skipping of †controlled:statements. 1a7a11

5.8 †Loop:Statement, †Test:Statement 1a8

The †loop:statement provides for the iteration of a †controlled:statement. 1a8a

.1 †For:clause is defined below. †Controlled:statement means the full generality of †statement under the control of iteration management mechanisms. Any †statement then can be iteratively operated in a †loop:statement. If the †controlled:statement starts with ←FOR or ←BEGIN FOR, this is a nested †loop:statement. Multiple (parallel) control is provided with only one †for:clause. 1a8a1

.2 The iterations or repetitions of the †controlled:statement are managed by means of one or more †control:variables which are established and maintained within the †loop:statement. The †loop:statement

consists, then, of a means of specifying and controlling †control:variables and a †controlled:statement that is to be iteratively operated. 1a8a2

.3 A †letter:control:variable, represented by a single †letter, is introduced within the †loop:statement for the purposes of iteration control, and is defined or active only within the immediate †loop:statement. See Section 5.10 for a more complete discussion of the scope of a †letter:control:variable. 1a8a3

.4 From the definitions above, it is seen that a †for:clause may have a list of †loop:controls (for parallel control) where each †loop:control consists of a †control:variable followed by a list of †control:clauses enclosed in †parentheses. The †control:clauses associated with each †control:variable provide the successive values to be assigned to that associated †control:variable for successive executions of the †controlled:statement. Each †control:variable is given a successor value for each execution of the †controlled:statement. Execution of the †loop:statement is terminated when the †controlled:statement causes a non-return jump out of the †loop:statement or when, for any one of the †loop:controls, there is no successor available. The leftmost †control:clause in each parenthesized list is used first to assign values to the †control:variable. When it has been fully utilized, the next †control:clause in the list provides values. Utilization of the †control:clauses proceeds in this manner until the list is exhausted. 1a8a4

.5 Each †control:clause as defined above consists of up to three parts or phrases to specify the range for iterative operation of the †controlled:statement. 1a8a5

a. The †initial:phrase, if present, must come first in the †control:clause and serves to provide an initial value for the †control:variable. 1a8a5a

b. There may be either a †replacement:phrase (introduced by ←THEN to specify the next value for the †control:variable or an †increment:phrase (introduced by ←BY to specify the amount by which the †control:variable is to be modified on each iteration. 1a8a5b

c. A †terminator:phrase (introduced by ←UNTIL or ←WHILE may contain the test by which the end of the iteration process is determined. 1a8a5c

.6 During repeated cycling within a †control:clause, †formulas are normally reevaluated during each cycle. However, if the syntax permits and the †formula is enclosed in †brackets (indicating a †value:formula), the †value:formula is evaluated once upon entering the †control:clause and that value is used repeatedly without change until the †control:clause is exited. †Value:formulas in the †replacement:phrase, †increment:phrase and †terminator:phrase are evaluated after the †control:variable has been once given the value from the †initial:phrase, if there is one. The value (or even existence) of a †letter:control:variable and of †value:formulas are undefined if the †controlled:statement is entered via a jump to an internal †statement:name. 1a8a6

.7 In the general scheme, iteration begins at some initial value and continues in increments of a certain amount until (or while) a specified condition is detected. However, all of the parts of a †control:clause are optional except that a †letter:loop:control must have an †initial:phrase in its first †control:clause unless this same †letter is used as the †control:variable of a †loop:statement containing this one (nested). The effects of omitting various parts of the †control:clause are detailed in the table below. 1a8a7

.8 In utilizing the †control:clauses in accordance with the above table, every †formula is normally evaluated each time it is used. However, a †value:formula is evaluated only the first time it is encountered in executing the †loop:statement and the value is saved for subsequent use. When a value is added as stated in the table, its algebraic sign is, of course, taken into consideration. The presence of a †terminator:phrase causes testing after the †control:variable gets its new value (if it does get a new one) and before the †controlled:statement is executed--indeed, before the next †loop:control is attended to. The termination mentioned in the table applies really to utilization of the †control:clauses. If the †primitive is ←WHILE and the †conditional:formula is ←0 (false) or if the †primitive is ←UNTIL and the †conditional:formula is ←1 (true), instead of going on to the next †loop:control or executing the †controlled:statement immediately, the next †control:clause is utilized--or the †loop:statement is terminated if there are no more †control:clauses to be utilized with this †loop:control. In cases 1A, 1B, 3A, 3B, 4A, and 4B above, the previous value is whatever is

left by previous operations including, if it happens, the value left by the previous †control:clause but not used in an execution of the †controlled:statement because of the condition of the terminator. 1a8a8

.9 It bears repeating--for every execution of the †controlled:statement a value is specified for each †control:variable by its respective associated list of †control:clauses. Whenever any one of the several lists of †control:clauses has been fully utilized and there is no value available for its respective †control:variable, execution of the †loop:statement is terminated. 1a8a9

.10 Consider an †indexed:variable as a †control:variable. In this case, a particular instance of the referenced †indexed:variable must be considered as well as the iteration of the †controlled:statement. It is possible for one iteration of a †controlled:statement to be performed based on a certain instance of a †named:variable as the †control:variable while the next iteration employs a different instance of that †indexed:variable. Each time an †indexed:variable as a †control:variable is to receive a new value from its associated list of †control:clauses, its †index is evaluated--to provide the particular instance of the †control:variable to be used. If incrementation is indicated, it is the presently indicated instance of the †control:variable that is incremented. The incrementation is performed as if by an †assignment:statement in which the old value is incremented and returned to the †indexed:variable. The rules affecting the order of evaluation of elements are in effect and the programmer is responsible for avoiding undesirable side effects. If the new value is merely the old value, without replacement or incrementation (first use in cases 1,3,and 4), all values are left undisturbed. In this last situation, there may not even be a need to evaluate the †index at this point. 1a8a10

.11 The †test:statement is permitted only within a †loop:statement. If a †control:variable is referenced, it must be a †control:variable for a †loop:statement (possibly nested) within which the †test:statement appears. Since all incrementing, replacing, and testing occurs before execution of the †controlled:statement, the †test:statement will invoke the †loop:controls for the designated †control:variable and all those which †follow it rather than those which precede it in the †for:clause. 1a8a11

.12 Execution of the †test:statement transfer execution to the †loop:controls in the †for:clause at the top of the †loop:statement--whichever †control:clauses are active at the moment. If no †control:variable is referenced, transfer is to the first †loop:control; and (in case the loops are nested) †loop:controls for the innermost loop of the nest in which the †test:statement appears are invoked in the order in which they occur in the †for:clause. 1a8a12

.13 If a †control:variable is referenced in a †test:statement, transfer is to the †loop:control associated with the referenced †control:variable--skipping earlier parallel †loop:controls in the same †for:clause, and (in case the same †variable is a †control:variable in more than one †for:clause for nested †loop:statements) to the innermost active †loop:control associated with referenced †control:variable. 1a8a13

5.9 †Loop:Statement Execution 1a9

The †control:clauses associated with each †control:variable may be considered to form a list with a pointer indicating which clause to utilize. The pointer may even have two heads, the first pointing to the initial value (or the place reserved for the initial value) and then to the replacement or the increment and the second pointing to the terminator. The descriptions of what happens with regard to the various kinds of †control:clauses indicate what happens to the pointers. They remain with the clause just utilized or they go on to the next--the first pointer head moves from the initial value to the replacement or increment, or it remains with the replacement value or the increment. 1a9a

.1 The position of each pointer is undefined until the †loop:statement begins execution the first time. At that moment all pointers for the †loop:controls in that †loop:statement are set to point to their respective first †control:clauses. This happens every time upon normal entry to the †loop:statement through the top. 1a9a1

.2 From that time on, as long as the scope containing the †loop:statement remains active, the positions of the pointers are defined. If control leaves the loop through execution of a †procedure:call:statement or a †go:to:statement, the positions of the †control:clause pointers are undisturbed. If execution of the loop is resumed (such as by means of a †go:to:statement

referencing a †statement:name within the †controlled:statement), utilization of the †control:clauses proceeds as if the †loop:statement had not been interrupted. 1a9a2

.3 Each †loop:control may be considered to have an additional clause that says, in effect, "terminate now". If the pointer is moved to this implicit clause and the clause is then examined, execution of the †loop:statement terminates immediately; no action is taken with regard to subsequent †loop:controls in case there are more than one. The pointer will continue to point to the "terminate now" clause and will not be moved again until it is initialized through a subsequent normal execution of the †loop:statement. If now, the loop is entered "through a side door" it will surely terminate when the iteration mechanisms is invoked, regardless of what values the †control:variables may have (perhaps set outside the loop) because one pointer is pointing to "terminate now". Before termination, however, †control:variables which occur earlier in the †loop:statement than the one which caused the previous termination will be incremented or whatever. In fact, one of these earlier †control:variables may now cause the termination and there will then be two pointers pointing to their respective "terminate now" clauses. Of course, if the iteration mechanism is invoked by means of a †test:statement that bypasses all †loop:controls pointing to "terminate now", the †controlled:statement may be executed again. 1a9a3

.4 An example of a parallel †loop:statement is: 1a9a4

```
.A1 = 0;                                          1a9a4a
.FOR A1(1 THEN 1-A1 UNTIL 0)                      1a9a4b
.B2(1 BY 1 UNTIL C3 = 8)                          1a9a4c
.C3(1,1,2,3,5);                                   1a9a4d
.S4: D5[B2] = A1 * C3;                            1a9a4e
.C3 = C3 + 3;                                     1a9a4f
.IF C3 < 12; S4; STOP;                            1a9a4g
```

.5 In order to illustrate the flow of execution for the example above, the diagram below illustrates the changing

of values for the †variables in the example. If there is a number at the intersection of arrow and a column, the †variable at the left of the row receives that value. The setting of values proceeds down the leftmost column, then down the column to its right, etc. 1a9a5

	←A	B	C	D	E	F	G	H	J	K	L	M	
A1	0	1	0	1	0	1	0		1		0		1a9a5b
B2		1	2	3	4	5	6		7				1a9a5c
C3		1	1	2	3	5		8		11		14	1a9a5d
D5[1]		1											1a9a5e
D5[2]			0										1a9a5f
D5[3]				2									1a9a5g
D5[4]					0								1a9a5h
D5[5]						5							1a9a5i
D5[6]							0						1a9a5j
D5[7]									11				1a9a5k

.6 Column ←A above represents the setting of ←A1 by the first †statement in the example. Column ←B represents the initialization of the †control:variables and the first execution of the †controlled:statement. Columns ←C, ←D, ←E, and ←F represent iterations of the loop. In column ←G, we have another in the series of replacements for ←A1, another incrementation of ←B2, and termination because the next clause for ←C3 is "terminate now". Column ←H shows the setting of ←C3 outside the loop and the setting of ←D5[6] because of the jump into the loop. Column ←J shows another replacement for ←A1, incrementation of ←B2, and termination--this time because the test on ←C3 (in the ←B2 control) turns out "true". Column ←K shows a new setting for ←C3 outside the loop and the setting of ←D5[7] because of the jump back in at ←S4. Column ←L shows the replacement for ←A1 and termination--now because the ←B2 control points to "terminate now". Column ←M shows the setting of ←C3 outside the loop. The program now stops because the test on ←C3 fails. 1a9a6

5.10 Scope of †Letter:Control:Variables 1a10

The scope of a †letter:control:variable is just the †loop:statement in which it is defined and activated. A †letter:loop:control defines the †letter:control:variable and it is assigned the value of the †initial:phrase of its first †control:clause. The †letter:control:variable is then active and may be used as an †integer:variable until the end of the †controlled:statement of the †loop:statement. The †letter:control:variable may also be used in the †increment:phrase, †replacement:phrase, or †terminator:phrase of the first †control:clause of the †letter:loop:control that activates it and in any †formula of other †control:clauses of the †letter:loop:control. 1a10a

.1 In a nested †loop:statement, if an inner †letter:loop:control uses as a †control:variable a †letter that is already active as a †letter:control:variable, it is treated as a †named:variable in the inner †loop:statement. Its first †control:clause need not have an †initial:phrase. The †letter:control:variable is the same †variable as the one defined in the outer †loop:statement, and its final value in the inner †loop:statement is carried into the outer †loop:statement. 1a10a1

.2 The same †letter may be used as the †letter:control:variable for non-nested †loop:statements, but these †control:variables are then considered as different †variables. 1a10a2

.3 A †letter:control:variable remains active only so long as execution remains within the †loop:statement. In general, †letter:control:variables are deactivated whenever control is transferred outside the †loop:statement by means of a †go:to:statement or by coming out the bottom because of completion of the †loop:statement. †Letter:control:variables are not deactivated when a procedure or function is called and that procedure or function returns control to this †loop:statement. 1a10a3

.4 †Procedure:declarations and †switch:statements may occur inside a †loop:statement with †letter:loop:control and they may reference the †letter:control:variable. However, the value or even the existence of a †letter:control:variable is undefined if the †controlled:statement is entered via a jump from an outer scope to a †statement:name in the †loop:statement, or if

the internal procedure or function is called from outside the †loop:statement. 1a10a4

5.11 †Procedure:Call:Statement 1a11

A †procedure:call:statement is used to invoke a procedure that is not a function. 1a11a

.1 For a discussion of †remquo:procedure:call:statement, see Section 5.11.17. 1a11a1

.2 †Actual:input:parameters must match the †formal:input:parameters associated with the named procedure (or alternate entrance) in number, kind, and position in the list, and †actual:output:parameters must match the †formal:output:parameters in number and position in the list. The matching as to kind is that if the †formal:input:parameter is a †statement:name, the corresponding †actual:input:parameter must be a †statement:name or one of the forms beginning with ←STOP, ←RETURN, ←TEST, or ←EXIT. If the †formal:input:parameter is an †item:name, the corresonding †actual:input:parameter must be a †formula. If the †formal:input:parameter is a †table:name or a †data:block:name, the corresponding †actual:input:parameter must be a †data:block:name, a †table:name (with or without an †index), a †variable, or ←@ followed by a †pointer:formula. If the †formal:input:parameter is a †procedure:name, the †actual:input:parameter must be the †name of a procedure with the same number, kind, and position of †formal:input:parameters and †formal:output:parameters as the procedure used as a †formal:input:parameter. 1a11a2

.3 In a procedure making a call on a procedure which is one of its †formal:input:parameters, the conversions between †actual:input:parameters and †formal:input:parameters are made in accordance with the descriptions of the †formal:input:parameters of the procedure as a †formal:input:parameter. No cognizance is taken of the descriptions of the †formal:input:parameters of the procedure which is given as an †actual:input:parameter. Consider this example: 1a11a3

```
←PROC AA (BB,CC);                                   1a11a3a

←BEGIN PROC BB (DD);                                1a11a3b
```

```
procedure ←AA          BEGIN ITEM DD F; END    procedure
←BB (formal †parameter)                                    1a11a3c

←ITEM CC U 26;                                             1a11a3d

←BB (CC);                                                  1a11a3e

←END                                                       1a11a3f

←ITEM COUNT S 15;                                          1a11a3g

←PROC EE (FF);                                             1a11a3h

←BEGIN ITEM FF U 30;                                       1a11a3i

←COUNT = COUNT + FF;              procedure ←EE (actual
†parameter)                                                1a11a3j

←END                                                       1a11a3k

←AA (EE,5);                     the executed †statement    1a11a3l
```

.4 In the example above, the value ←5 is assigned to ←CC, using whatever conversion is called for by the assignment rules, during invocation of ←AA. Then the formal invocation of ←BB causes ←CC to be converted as if for assignment to ←DD (there need be no actual space allocated for ←BB or ←DD. There is †no further conversion from ←DD to ←FF--it is the programmer's responsibility to see that ←FF matches ←DD (or to accept the consequences if it does not). 1a11a4

.5 The order of evaluation of parameter data is left to right. The values of †actual:input:parameters are assigned to †formal:input:parameter:variables from left to right as if by an †assignment:statement (Section 5.5). Upon exit, †formal:output:parameters are assigned to †actual:output:parameters from left to right as if by an †assignment:statement. This order of assignment is certainly of significance if some †formal:input:parameters are pointers to other †formal:input:parameters or if some †actual:output:parameters are pointers to other †actual:output:parameters. 1a11a5

.6 If a procedure is to refer to an external table or data block, the location of the table or data block may be passed to the †formal:input:parameter which will be used by the procedure as the pointer in its references to

its local table or data block. The location of the external structure can be indicated by using a †location:function:call or a †pointer:formula evaluating to the correct location. The procedure will then utilize the pointed-to space in accordance with the declared structure of its local pointed-to table or data block. Alternatively, a local †table:name or †data:block:name can be a †formal:input:parameter. In that case, the †actual:input:parameter represents allocation that will become the value of the pointer (either programmer designated or compiler supplied) to the local structure. It doesn't matter if the †formal:input:parameter is a table or a data block. If the corresponding †actual:input:parameter is a †variable, a †table:name, or a †data:block:name, the location of the variable, table , or datablock (as if called forth by the use of ←LCC) becomes the value of the internal pointer. This location is the address of the word in which the variable or other data structure begins. No bit or byte locating information within the word is involved. If it is desired to provide a †formula (perhaps just a †constant or †variable) as a location value to be provided to the internal pointer to table or data block used as a †formal:input:parameter, the †formula must be preceded with an ←@. 1a11a6

.7 Note that †table:name or †variable has a different meaning as an †actual:input:parameter depending on the corresponding †formal:input:parameter. If the †formal:input:parameter is a †table:name or †data:block:name, a †table:name or †variable as an †actual:input:parameter means the location (of the variable or table). If the †formal:input:parameter is a †variable, a †table:name as an †actual:input:parameter means the value of the first †entry:variable, and any other †variable as an †actual:input:parameter just means its value. 1a11a7

.8 Consider, for example, a procedure that processes messages in the form of long †character:variables. Within the †procedure:declaration a table (←T1) is declared with one entry consisting of a very long †character:variable 8←(C1). ←T1 is also a †formal:input:parameter. Therefore, no space is allocated for the table or its item. Now the †procedure:call:statement uses the †name of a †character:variable (←MSG) as the corresponding †actual:input:parameter. The location of ←MSG is passed in to the procedure and becomes the value of the pointer

(named or unamed) to ↓T1. If the procedure looks at one message and creates another, perhaps two such †formal:input; and †actual:input:parameters are needed. 1a11a8

.9 If the procedure exits by means of a †go:to:statement referencing a †formal:input:parameter, the first effect is to set the †actual:output:parameters from the values of the †formal:output:parameters. The next effect depends on the nature of the †actual:input:parameter corresponding to that †formal:input:parameter. If the †actual:input:parameter is a †statement:name, it is as if a †go:to:statement referencing that †statement:name were executed at a point immediately following the †procedure:call:statement. If the †actual:input:parameter is one of the forms beginning with ↓STOP, ↓RETURN, ↓TEST, or ↓EXIT, it is as if a corresponding †stop:statement, †return:statement, †test:statement, or †exit:statement were executed at a point immediately following the †procedure:call:statement. 1a11a9

.10 A †go:to:statement referencing an outer †statement:name causes a jump to the point named and it also deactivates all procedures called from the scope of that outer †statement:name, and procedures called by those procedures, etc. It bypasses the setting of †actual:output:parameters of the procedure in which the †go:to:statement is executed and all other procedures deactivated by the jump. If, however, the outer †statement:name is a †formal:input:parameter in its own scope, the †actual:output:parameters of that procedure re set before control is transferred in accordance with the †actual:input:parameter corresponding to that †statement:name used as a †forsal:input:parameter. 1a11a10

.11 The deactivation through execution of a †go:to:statement can occur recursively. Suppose that ↓ALPHA is a recursive procedure in which the procedure ↓BETA is nested and te procedure ↓GAMMA is nested in ↓BETA. Suppose also that ↓ALPHA had been invoked, it then called ↓BETA, which called ↓GAMMA which called ↓ALPHA (creating a second copy of ↓ALPHA, including ↓BETA and ↓GAMMA. Now, once again ↓ALPHA calls ↓BETA and ↓BETA calls ↓GAMMA. If the second copy of ↓GAMMA jumps directly to a †named:statement in ↓ALPHA, it is a jump to that †statement in the second copy (or perhaps second use in case ↓ALPHA is reentrant). It deactivates the second activations of ↓BETA and ↓GAMMA but not the first. 1a11a11

.12 There are several attributes of procedures that can lead to complications in implementation and execution. The least traumatic attribute is a pointed-to data space. This can be implemented by the calling program placing the value of the pointer in a base register which is then used by the procedure in all its references to its private data. 1a11a12

.13 Complications due to the use of external procedures depend on the amount of work done by the loader. If the loader resolves the locations of all †formal:input:parameters and †formal:output:parameters as well as of the procedure, no further complication arises. If the loader does not resolve these locations, then it is necessary for the calling program to convert †actual:input:parameters, if necessary, and store them or their locations in a standard communications area from whence they are retrieved by the called program. The process is reversed for †actual:output:parameters. 1a11a13

.14 For recursive procedures, it becomes necessary to save some extra pointer values in the data space assigned for each †procedure:call:statement in order to be able to unwind. The †recursive:directive (Section 11.7.5) may be important in this regard. If procedure ←P1 calls procedure ←P1, the †recursive:directive is unnecessary--it is obvious to the compiler that ←P1 is a recursive procedure. On the other hand, if ←P2 calls ←P3 and ←P3 calls ←P2, recursiveness depends on information not available to the compiler. It may happen that execution is such that there is never more than one each active call of ←P2 and ←P3. But if ←P3 does actually call ←P2 after it has been called by ←P2, then the programmer uses the †recursive:directive to indicate that ←P2 is recursive. 1a11a14

.15 Whether and where the †recursive:directive is needed is system dependent. Recursive procedures will probably be able to unwind properly if all procedures with pointed-to data space implement calls on other such procedures in a manner similar to the following: 1a11a15

a. Call on procedure with pointed-to data space from within procedure with pointed-to data space (←r1 and ←r2 are registers set aside for such calls):

Conditions: ←r1 points to current data space.
←r2 points to data space of procedure that called this one. 1a11a15a

b. This procedure calls a procedure: 1a11a15b

```
←r2      ←SAVE @ r1                                    1a11a15b1

←r1      ←r2                                           1a11a15b2

↑pointer:formual (elements ←@ r2  evaluated      ←r1
                                                       1a11a15b3
↑actual:input:parameters ←@ r2
↑formal:input:parameters ←@ r1                         1a11a15b4

return jump (called procedure saves return ←@ r1
and jumps back)                                        1a11a15b5

↑formal:output:parameters ←@ r1
↑↑actual:output:parameters ←@ r2                       1a11a15b6

←r2      ←r1                                           1a11a15b7

←SAVE @ r↑      ←r2                                    1a11a15b8
```

.16 The following example is a simple recursive procedure that calculates the factorial of a number. It is certainly not the recommended way to calculate a factorial, but it illustrates recursion: 1a11a16

```
←PROC FCTRL @ ( G1 : G2) ;                             1a11a16a

←BEGIN ITEM G1, G2, Q1 U ;                             1a11a16b

←G2 = 1 ;                                              1a11a16c

←IF G1 <= 1 ; RETURN ;                                 1a11a16d

←Q1 = SPACE ( DSIZE ( FCTRL) ) ;                       1a11a16e

←FCTRL @ Q1 (G1 - 1 : G2 ) ;                           1a11a16f

←GARBAGE (Q1) ;                                        1a11a16g

←G2 = G2 * G1 ;                                        1a11a16h

←END                                                   1a11a16i
```

.17 The ↑remquo:procedure:call:statement is used to obtain the quotient and remainder that result from dividing the first ↑actual:input:parameter by the second ↑actual:input:parameter. 1a11a17

.18 The compiler may implement the †remquo:procedure:call:statement by efficient inline code (preferably) rather than a call to a procedure. The effect of this code is as if the following procedure were called: 1a11a18

```
←PROC REMQUO ( NUM, DEN : QUO, REM) ;                    1a11a18a
←BEGIN ITEM NUM, DEN, REM, QUO S n;                      1a11a18b
  ←QUO = NUM/DEN ;                                       1a11a18b1
  ←REM = NUM - QUO * DEN ;                               1a11a18b2
←END                                                     1a11a18c
```

.19 The †item:declaration above contains the letter ←n, which stands for a system-dependent size for these integer items. It will undoubtedly be the maximum size for convenient arithmetic with integers rather than the size used for pointers and addresses, which is likely to be the system default size. 1a11a19

5.12 †Go:To:Statement, †Stop:Statement, †Return:Statement, †Exit:Statement 1a12

Normally, the †statements of a †processing:declaration are executed in the order in which they are written. The four †simple:statements below are used in modifying this normal execution order of the †statements. The †test:statement (Section 5.8.12) discussed in connection with the †loop:statement of which it may be a part, the †procedure:call:statement (Section 5.11), and the †switch:statement (Section 5.13) are also used to control the execution order of †statements. 1a12a

.1 The †go:to:statement effects a transfer of control to the †statement bearing the referenced †statement:name. A single-component †index is permitted only in transferring control to a named †switch:statement. In such a reference, the value of †index is used instead of the value of the †numeric:formula beginning the †switch:statement in selecting the constituent †statement to be executed. If a †go:to:statement referencing a †switch:statement has empty †index †brackets, it is as if there were an †index with the lowest meaningful value with respect to the †switch:statement. A †go:to:statement with an †index (or empty †index †brackets) may reference a †statement:name that precedes

the .BEGIN of a !switch:statement. In fact, a !go:to:statement, without !index or !brackets, referencing such a !statement:name has the effect of a !go:to:statement with empty !brackets. The vaue of the !index (or its lowest meaningful value in case it is omitted) is used to select the constituent !statement to be executed. 1a12a1

.2 The !stop:statement is the logical termination of execution of a program. Depending on the system, .STOP may cause a machine halt or a normal return to the executive. 1a12a2

.3 The !return:statement is permitted only within a !procedure:body. Its effect is to terminate execution of the procedure, set the !actual:output:parameters from the !formal:output:parameters, and return control to the !statement following the call in whatever program invoked the procedure. The call might have been in any scope such as another procedure, the main program, or even the system executive. 1a12a3

.4 It is immaterial whether the !return:statement references a !procedure:name or an !alternate:entrance:name; the return will be that associated with the active entrance in any case. If no !name is referenced, the !statement means to return from the most local procedure. After the last !statement in a !procedure:body is executed, if it does not transfer control, return from the procedure is effected as if the !statement .RETURN; had been executed. 1a12a4

.5 Within nested procedures, the referenced !name in a !return:statement means to return from the procedure having the referenced !name as its normal or alternate entrance. (If nested procedures use the same !name, it means return from the most local procedure, within which the !statement appears, having the referenced !name.) If return is made for an outer procedure from within an inner procedure, the !actual:output:parameters are not set for the inner procedure. 1a12a5

.6 Return to a procedure that is not active is undefined. "Active" means the procedure has been called but an explicit or implicit return from the procedure has not yet been executed. Such a return could only be attempted from a procedure declared with an external definition within another procedure. 1a12a6

.7 There exists a school of programming philosophy that holds that the use of †go:to:statements leads to poor coding practices and renders certain optimization techniques impossible. Programming done in accordance with the precepts of this school is known as structured programming. Inclusion of the †exit:statement (and the †switch:statement) makes structured programming possible in JOVIAL. It is not suggested that any compiler ought to, but it would certainly be possible to build a compiler to enforce structured programming or to issue a warning when the precepts of structured programming are violated. 1a12a7

.8 Heretofore, a †statement:name was considered only to designate a point, an entrance point, in a program. To give meaning to the †exit:statement, a †statement:name must be understood to have reference to an entire program structure--the †statement to which it is applied, its entrance point, and its exit point. An †exit:statement may only appear between the entrance point and the exit point of the program structure whose †name it references. The effect of executing the †exit:statement is to transfer control to the exit point of the program structure. The interstices between program structures smetimes have a fine structure so that, for example, the exit point of a †loop:statement is not the same as the exit point of its †controlled:statement. 1a12a8

.9 The effect of an †exit:statement referencing a †name attached to a †procedure:body is the same as a †return:statement for that procedure. The effect of an †exit:statement referencing a †name attached to the †controlled:statement of a †loop:statement is the same as a †test:statement, without reference to a †control:variable, in the †named:statement but not in any loop nested within that †named:statement (even if the †exit:statement is in the nested loop). The effect of an †exit:statement referencing a †name attached to a †loop:statement terminates execution of the loop. The effect of an †exit:statement referencing a †statement:name that precedes the BEGIN of a †switch:statement is the same as if that †statement:name were attached to the †switch:statement. 1a12a9

.10 The following example illustrates the details of the effect of executing various †exit:statements. The relevant program details and the †exit:statements are on the left. The notation . . . indicates a sequence of

```
!statements.  To the right of each !exit:statement is an
equivalent !statement and an explanation.                    1a12a10

  .L0:  BEGIN                                                1a12a10a

  .     . . .                                                1a12a10b

  .     EXIT L0                GOTO L1;          (Exit
from !compound:statement)                                    1a12a10c

        . . .                                                1a12a10d

        END                                                  1a12a10e

  L1:   FOR I (1 BY 1 UNTIL I = 9);                          1a12a10f

  L2:   BEGIN                                                1a12a10g

        . . .                                                1a12a10h

        EXIT L1;               GOTO L3;          (Exit
from the !loop:statement)                                    1a12a10i

        . . .                                                1a12a10j

        EXIT L2:               TEST;             (Exit
from the !controlled:statement)                              1a12a10k

        . . .                                                1a12a10l

        END                                                  1a12a10m

  L3:   IF  ALPHA > 0;                                       1a12a10n

  L4:   BEGIN                                                1a12a10o

        . . .                                                1a12a10p

        EXIT L3;               GOTO L6;          (Exit
from the !conditional:statement)                             1a12a10q

        EXIT L4;               GOTO L6;          (Exit
from !controlled:statement                                   1a12a10r

        . . .                                      is
effectively the same)                                        1a12a10s

        END                                                  1a12a10t
```

```
L5:  ELSE                                                    1a12a10u

     BEGIN                                                   1a12a10v

     . . .                                                   1a12a10w

     EXIT L5;                GOTO L6;          (Exit
from †controlled:statement)                                  1a12a10x

     . . .                                                   1a12a10y

     END                                                     1a12a10z

L6:  SWITCH ALPHA + 1;                                       1a12a10a@

L7:  BEGIN                                                   1a12a10aa

L8:  BEGIN                                                   1a12a10ab

     . . .                                                   1a12a10ac

     EXIT L6;                GOTO L11;         (Exit
from †switch:statement)                                      1a12a10ad

     EXIT L7;                GOTO L11;         (Exit
from †switch:statement)                                      1a12a10ae

     . . .                                                   1a12a10af

     EXIT L8;                GOTO L11;         (Not
←L9, because of the                                          1a12a10ag

     . . .                                      †comma
before ←L9                                                   1a12a10ah

     END                                                     1a12a10ai

L9:  BEGIN                                                   1a12a10aj

     . . .                                                   1a12a10ak

     EXIT L9;                GOTO L10;         (Exit
from †compound:statement--                                   1a12a10al

     . . .                                      no
†comma follows)                                              1a12a10am

     END                                                     1a12a10an
```

```
L10:  . . .                                                    1a12a10ao
      END                                                      1a12a10ap
L11:  . . .                                                    1a12a10aq
      PROC P1;                                                 1a12a10ar
L12:  BEGIN                                                    1a12a10as
      . . .                                                    1a12a10at
      PROC P2 ;                                                1a12a10au
L13:  BEGIN                                                    1a12a10av
      . . .                                                    1a12a10aw
      EXIT L12;              RETURN P1;          (Exit
from †procedure:body)                                          1a12a10ax
      . . .                                                    1a12a10ay
      END                                                      1a12a10az
      . . .                                                    1a12a10b@
      END                                                      1a12a10ba
```

5.13 †Switch:Statement 1a13

A †switch:statement provides a multipath branch to other †statements contained within it. 1a13a

.1 Each †statement between ←BEGIN and ←END in the above form is associated with an integer. In the absence of explicit bracketed †numbers ahead of or between †statements, the first †statement is associated with zero and successive †statements (including †null:statements) are associated with successive integers. (A †compound:statement, †switch:statement, †loop:statement, or †conditional:statement counts as a single †statement.) Each bracketed †number, where present, interrupts the succession of associated integers and states a positive or negative integer value to be associated with the next †statement. The succession then resumes following the stated value. There must be no repetition in values--each †statement must be associated with a unique integer value. The †statements and their associated

integer values are then effectively reordered so that the integer vaues are in monotonically increasing order, with no duplications but possible gaps. Some of the †statements may be followed by †commas. 1a13a1

.2 In executing the †switch:statement, the †numeric:formula following .SWITCH is evaluated as an integer (truncated if necessary). Then the †statement enclosed in the .BEGIN END brackets and corresponding (as described above) with the values of the †formula is executed. If the †numeric:formula does not yield a value corresponding to a †statement in the list (including †null:statements), the result is undefined. Values skipped due to explicit †numbers in the list do not correspond to †statements. A †statement in the list can be executed, if it bears a †statement:name, by execution of a †go:to:statement somewhere that references that †name. 1a13a2

.3 After execution of any †statement in the list, if it does not permanently transfer execution elsewhere, the effectively next †statement in the list is executed, unless they are separated by a †comma or a gap in the sequence of integer values, in which case the execution sequence is transferred to the †statement following the .END. 1a13a3

.4 Execution control can arrive at a †switch:statement in three ways: by "falling through" from the †statement preceding the †switch:statement, through execution of a †go:to:statement referencing the †name of the †switch:statement and without an †index or †index †brackets, or through execution of a †go:to:statement that does have a one-component †index and references the †name of the †switch:statement. The first two of these ways result in the execution of the †switch:statement as described in Section 5.13.2. 1a13a4

.5 The use of the †index in a †go:to:statement referencing a †name before the †switch:statement (or a †name before the .BEGIN of the †switch:statement even if there are no †index and †brackets) means that the †numeric:formula of the †switch:statement is not evaluated. Instead the value of the †index with the †go:to:statement is used to select the †statement in the list to be executed. Evaluation of the †numeric:formula is omitted even if an †order:directive precedes the †switch:statement. If the †go:to:statement has empty †brackets for the †index, it is as if there were an

↑index present having the smallest value associated with a ↑statement in the list. 1a13a5

.6 The following example and flow diagram illustrate the use of a ↑switch:statement: 1a13a6

```
←SW:  SWITCH ALPHA;                                    1a13a6a

      BEGIN [1] BETA = 3; ;                            1a13a6b

                GAMMA = BETA;,                         1a13a6c

                IF GAMMA = 2 ; BETA = 2;               1a13a6d

            [6] BETA = GAMMA;                          1a13a6e

      END       ALPHA = 7;                             1a13a6f
```

.7 In the example above, values of ←2 and ←3 for ←ALPHA cause the same path to be taken because the ↑statement corresponding to the value ←2 is a ↑null:statement. The ↑statement 1a13a7

```
←GAMMA = BETA;                                         1a13a7a
```

exits to the end of the ↑switch:statement because it is followed by a ↑comma. The ↑conditional:statement 1a13a8

```
←IF GAMMA = 2;  BETA = 2;                              1a13a8a
```

exits to the end of the switch because it is ↑statement number ←4 and there is no ↑statement number ←5 1a13a9

.8 Using the same example in Section 5.13.6, the ↑go:to:statement 1a13a10

```
←GOTO  SW  [BETA];                                     1a13a10a
```

would cause the value of ←BETA to be used rather than the value of ←ALPHA in executing the ↑switch:statement. The ↑go:to:statement 1a13a11

```
←SW  [   ];                                            1a13a11a
```

would cause the path for ←ALPHA equal to ←1 to be taken. 1a13a12

.9 If the optional ↑statement:name after the ↑numeric:formula of a ↑switch:statement does indeed occur and is referenced in an ↑exit:statement, it is as if the

†statement:name were attached to the †switch:statement and it causes an exit from the †switch:statement. 1a13a13

5.14 †Direct:Statement 1a14

The †direct:statement is a †simple:statement. It is used as a means for breaking out of the JOVIAL language within a program and writing some instructions in another language more directly related to the organization of the computer for which the program is being compiled. 1a14a

.1 What is legal and meaningful with a †direct:statement depends on the system. A †direct:statement may reference a JOVIAL †name in the same scope. The †name translates to a location, but the exact meaning of "location" is system dependent. A †sets:directive and a †uses:directive may appear immediately after the †primitive .DIRECT to inform the compiler of data elements referenced in the †direct:statement. 1a14a1

.2 If the optional †semicolon occurs after the †primitive .JOVIAL, it serves only as a terminator for the †direct:statement and is not a †null:statement. 1a14a2

.3 While machine-language code might in some cases be desirable for object program efficiency, there are obvious disadvantages to using †direct:statements. Errors that might be detected by the compiler if JOVIAL were used are more likely to go undetected. †Program:declarations containing †direct:statements are more difficult to transfer to another machine. 1a14a3

JOVIAL Manual--Chapter 5

(J30205) 9-MAR-74 14:33; Title: Author(s): Duane L. Stone/DLS;
Distribution: /RJC; Sub-Collections: RADC; Clerk: DLS;
Origin: <STONE>C5.NLS;1, 9-MAR-74 13:21 DLS ;

DLD2 11-MAR-74 13:06 30206

Natrch 11 Notice on WWDMS Testing at Gunter AFB

Kindly give this your immediate attention

Matrch 11 Notice on WWDMS Testing at Gunter AFB

Reference AFDSDC Msg 082100Z Mar 74 and NLS files <daughtry>Mar06-wwdms-plan, and <bergstrom>wwdms-dfb. 1

On Friday 8 March 1974 Dave Daughtry and Deane Bergstrom talked to Capt Cecil Martin of the AFDSDC. The following items were discussed: 2

a) AFDSDC request for two RADC people to be on site at Gunter AFB to assist in planning and conducting tests of the FILE MAINTENANCE portion of WWDMS, to include: 2a

1) RADC assistance in the establishment of plans and test procedures. 2a1

2) RADC help in evaluation and write-up of the test results relating to the File Maintenance system. 2a2

b) AFDSDC desire for the RADC/ISI "focal point" to be on site on 21 March .... Bergstrom to serve in this capacity due to other TDY for Daughtry. 2b

c) Assistance from other Ccmmands for coding required tests. 2c

d) Assistane from other Commands as test participants or test observers. 2d

e) Formal testing to commence on 25 March 1974. 2e

In this respect, a meeting was called by ISIM of affected personnel for 0830 hours 12 March to establish RADC/ISI plans for supporting the WWDMS test effort in the File Maintenance area. The following individuals have been identified as potential support personnel for the Gunter AFB tests and may spend a certain amount of time at the test site. 3

	name	week	
			3a
a)	Bergstrom	21 - 29 March	3b
b)	Liuzzi	25 - 29 March	3c
c)	Wingfield	1 - 12 April	3d
d)	Daughtry	4 - 12 April	3e

DLD2 11-MAR-74 13:06 30206

Matrch 11 Notice on WWDMS Testing at Gunter AFB

(J30206) 11-MAR-74 13:06; Title: Author(s): David L. Daughtry/DLD2;
Distribution: /DFB MAW RAL FJT RFI RFI; Sub-Collections: NIC; Clerk:
DLD2;
Origin: <DAUGHTRY>MAR11-WWDMS-TESTING.NLS;1, 11-MAR-74 13:02 DLD2 ;

trip esd

<CARRIER>ESDTRIP.NLS;1, 11-MAR-74 07:51 RJC ; 1

(Serial) number: 027 1a

(Name)(s) of Traveler(s): Frank Tomaini
Richard Nelson 1a1

(Symbol): ISI 1a2

(Date) of Departure: 5 Mar 74 1a3

(Number) of days: 1 1a4

(Destination): ESD/MCI, Bedford, MA 1a5

(Purpose) of Trip: To Brief ESD on RADC Structured Programming
Contract with IBM. 1a6

(Person)(s) Contacted: Col Whitson/ESD/MCI 1a7

(Job) Order Number: 55500803 1a8

---------- 1b

(Contract) Number: N/A 1c

(Minutes) available?: No 1d

When?: 1d1

Where?: 1d2

(Follow) up Requirements?: No 1e

Date Required: 1e1

Responsible agency or individual: 1e2

Action Item: 1e3

(Summary) of events: 1f

IBM Project Engineer, John Naughton, briefed ESD/MCI and
supporting MITRE personnel on the Scope, Approach and
Deliverable items concerned with the RADC sponsored contract on
Structured Programming. 1f1

Of interest, were the facts that the contract will provide: 1f2

A Definition of Structured Programming. 1f2a

Documentation Requirements Training Materials and Job Descriptions for a typical programming effort. 1f2b

Initial Training Course for Air Force personnel. 1f2c

Material for early implementation of DAIS Software. 1f2d

Col Whitson (MCI), ESD Focal Point for the briefing, was asked to identify an ESD software candidate for implementation using Structured Programming technology. Col Whitson stated that it was the consensus of ESD/MITRE personnel that the effort was excellent in detail and in its objectives. It would serve as a baseline in the long climb toward achieving the goal of purchasing cost-effective software for the Air Force. He said it was a contract that would provide the Acquisition Division with a set of procurement tools for software. It appears to be the first major step in converting the act of producing software from an art to an engineering function. He further said that he would identify a candidate ESD Software Package. 1f3

RJC 11-MAR-74 17:22 30207

trip esd

(J30207) 11-MAR-74 17:22; Title: Author(s): Roberta J. Carrier/RJC;
Distribution: /FJT; Sub-Collections: NIC; Clerk: RJC;

JI 12-MAR-74 06:34 30209

FEEDBACK Identity Crisis

Susan:

I notice in your recent Journal Documents concerning your internal work on Feedback several statements that are in conflict with my understanding and would appreciate some clarification.

(1) You refer to FEEDBACK as the ident for the NLS Development Group with you as coordinator. Were you aware that at the USING meeting that the ident FEEDBACK was formed for the USING FEEDBACK Committe's work. How can this dual ident be explained?

(2) Additionally, the idents: NGRP [for netgripe] and NCMT [for netcomment] were formed. These idents have corresponding Initial Files in <USING> and are to be repositories for network feedback. A prototype system for the submission of network feedback is being debugged at Case-10. This system, using network mail, will make Journal submissions to these ident.

Your clarification of this apparent mistake in Idents would be most appreciated.

Jean

1

JI 12-MAR-74 06:34 30209

FEEDBACK Identity Crisis

(J30209) 12-MAR-74 06:34; Title: Author(s): Jean Iseli/JI;
Distribution: /SRL JCN FEEDBACK JOC(Jim, how's the prototype comming?)
NCMT; Keywords: feedback identity disambiguation; Sub-Collections:
FEEDBACK SRI-ARC; Clerk: JI;

WJC 11-MAR-74 21:03 30210

PURDUE LOGON problems with TSO

We have an account at CCN (ARP401, id:PAW, pass:MICLOG).
By the time you get this we may have found away around it but
at present it is impossible for us to logon to TSO because
we inadvertently CHANGED our logon PROC to a non-null
invalid procedure (whenever we attempt to logon we get
the message: 'LOGON' not found in command buffer...)

Could you somehow get us out of this rather sticky situation
by either (1) CHANGEing us back to NOPROC , or (2) telling us
how to substitute some existing procedure in the
LOGON PROC( ) command so we can get back on and do the
CHANGE ourselves?

Much thanks.

Bill Croft PURDUE@OFFICE-1

PURDUE LOGON problems with TSO

(J30210) 11-MAR-74 21:03; Title: Author(s): William James Croft/WJC;
Distribution: /BN; Sub-Collections: ; Clerk: WJC;
Origin: <PURDUE>MESS.NLS;1, 11-MAR-74 20:54 WJC ;

Questionnaire Response

Math 1244	1a1c7
Math 1250	1a1c8
Physics 1343	1a1c9
Elec 441T	1a1c10
Elec 442T	1a1c11
1968-69:B. Eng. IV	1a1d
Mech 562	1a1d1
Tech Paper 840	1a1d2
Math 1269	1a1d3
Math 1291	1a1d4
Elec 461T	1a1d5
Elec 462T	1a1d6
Elec 463T	1a1d7
Elec 464T	1a1d8
Elec 465T	1a1d9
Elec 466T	1a1d10
Elec 467T	1a1d11
Sociology 210	1a1d12
1969-70:B. Eng. V	1a1e
Elec 470	1a1e1
Elec 480T	1a1e2
Elec 482T	1a1e3
Elec 484T	1a1e4
Elec 485T	1a1e5
Elec 486T	1a1e6

Elec 494T	1a1e7
Elec 495T	1a1e8
Elec 496T	1a1e9
Elec 498T	1a1e10
Elec 499T	1a1e11
Philosophy 210	1a1e12
Carleton University	1a2
1970-71:(Partial)	1a2a
Contemporary Economic Issues	1a2a1
Advanced French Grammar	1a2a2
McGill University	1a3
1971-72:M.B.A. I	1a3a
Accounting	1a3a1
Behavioral Sciences	1a3a2
Computers and Systems	1a3a3
Finance	1a3a4
Marketing	1a3a5
Statistics	1a3a6
Operations Research	1a3a7
Industrial Management	1a3a8
Micro-Economics	1a3a9
Macro-Economics	1a3a10
1972-73:M.B.A. II	1a3b
Organizational Behavior	1a3b1
Skill Development	1a3b2

Management Policy	1a3b3
Research Paper(Computer Assisted Instruction)	1a3b4
Management in Public Sector	1a3b5
Corporate Planning	1a3b6
Finance	1a3b7
Statistics(Multivariate Analysis)	1a3b8
Statistics(Advanced Topics)	1a3b9
Operations Research(Linear Programming and Optimization)	1a3b10
Operations Research(Advanced Topics)	1a3b11
1973-74:(Partial)	1a3c
French Fifth Level Conversation	1a3c1
French Fifth Level Composition	1a3c2
Cost Benefit Analysis	1a3c3
Diplomas, Etc.	1b
B. Eng. (Electrical), 1970.	1b1
M.B.A. (Statistics and O.R.), 1973.	1b2
Member of Corporation of Engineers of Quebec, 1973.	1b3
Certificate of Proficiency in French (McGill U.), 1974.	1b4
Employment Experience	1c
Bell Canada	1c1
April 1973 to present	1c1a
published "Cross Impact Matrix Applications in Technology and Policy Assessment" and "Group Judgmental Data in Cross Impact Analysis and Technology Assessment".	1c1a1
Winter 1972-73	1c1b

published "A Technology Assessment of Computer Assisted Instruction in Colleges". 1c1b1

Summer 1972 1c1c

published "Internal and External Delphi Panel Comparison". 1c1c1

Dept. of Transport 1c2

May 1970 to August 1971 1c2a

Electrical Engineer 1c2a1

Summer 1969 1c2b

Hydrographic surveying 1c2b1

Dept. of Public Works 1c3

Summer 1968 1c3a

Land Surveying 1c3a1

Summer 1967 1c3b

Building Inspector 1c3b1

D. Rabin, Eng. 1c4

Summer 1966 1c4a

Land Surveying 1c4a1

Dept. of National Defense 1c5

Summer 1965 1c5a

Instructing trainees 1c5a1

Summer 1964 1c5b

Instruction techniques 1c5b1

Bell Canada Projects 1d

Liaison officer on Queen's U. study of impact of communications in North 1d1

Work on Cross Impact futures game with BC Tel	1d2
Work on Incasting with BNR	1d3
Monitoring Automatic Meter Reading	1d4
Evaluating Engelbart's NLS system	1d5
Work on evaluating conferencing with BNR	1d6
Work on relative energy costs of communications and travel	1d7
BPG Hudson Institute contact	1d8
Corporate Social Responsibility, Cost Benefit Paper	1d9
Technology Assessment Paper	1d10
Cross Impact Paper	1d11
Group Judgmental Data Paper	1d12
Organizations contacted	1e
Laval University	1e1
U of Montreal	1e2
McGill U	1e3
Sir George Williams U	1e4
U de Quebec	1e5
Queen's U	1e6
U of Windsor	1e7
U of Alberta	1e8
U of Melbourne	1e9
Stanford U	1e10
SRI	1e11
IFF	1e12
Hudson Institute	1e13

PF 12-MAR-74 08:07 30212

Questionnaire Response

(J30212) 12-MAR-74 08:07; Title: Author(s): Phil Feldman/PF;
Distribution: /PIW; Sub-Collections: NIC; Clerk: PF;

Section Pitch

TThe abiltiy of current computing to achiev this degree of flexabiltiy and responsiveness,it is the objective of theis program his is to be the outline of the pitch to be given to the front office in Febuary hopefully the week of the 18th. 1

Section-Systwems softwaee 2

It seem sto me that more and more what we are doing is trying to beef up,enlighnen users and bulid special tools so AF users can exploit and use the emerging tools systems offer such as text processing,data management. 2a

The AF ability to function as a modern day fightnig organization is becoming more and more dependent on computers.The posture of being able to operate in a limited war with various diploamatic restrictions is the overall current Nixon Doctirne.In order to achiev this goal a much more responsive command and control systems is required which can handle and process changing inouts from many other sources over a world wide network of computers which will include different types of information systems as well as in many cases different computers. 3

Section Pitch

(J30213) 12-MAR-74 09:06; Title: Author(s): Duane L. Stone/DLS;
Sub-Collections: RADC; Clerk: DLS;
Origin: <MCNAMARA>PROGRAM.NLS;2, 1-FEB-74 11:56 JLM ;

NIH Visit

Jo Naughton-Computer Facility Director 1

301-496-5381 1a

It turned out that he was involved with thh first full text search retrieval system developed at the U of Pittsburgh. 1b

The computer facility provides the data processing support to about 13,000 personnel within NIH. There are 12 research institutes with the bulk of the people on the grounds in Bethesda. It is a very large facility with 2

The first point of interest was that the facility is operated on a indutrial funding basis. He has no appropriated budget but has to sell his services to the institute researchers, who can go any where they want for their data processing support. 2a

He stated that this was no problem as most of his customers have been convinced that they get a better deal for the buck with him. He has subtle advantages, like he does not pay for the buliding. Things like this do effect the overall cost. He showed us a chart which he stated has been his basic tool for selling his operation. The chart show a curve which plots the cost of the average job over the past 70 year. It has went for $45 per average job to $5 per average job and he is convinced it will keep coming down. He had no qualms in pointing out that the average job was no fancy measure only the smallest to the largest and divided by the number of jobs. 2a1

He pointed out that one of the distinct advantages that they have experienced was that he got out of the position of being forced to decide which research project was the most important to run when he was close on money or time. A postion which he stressed was ridiculous, since what did he know about the priority etc. Now it is stictly on a cost basis. If a project runs out of computer money they go back to their boss for more, not to the faciltiy chief. 2a2

He further pointed out that it also got them out of the sad postion of competing for institute $ with all of the other directors and then ginving them back to the directors free. This way he did not antagonize the directors needlessly. They just pay as they go and it is a much cleaner way to operate. 2a3

DLS 12-MAR-74 09:08 30214

NIH Visit

(J30214) 12-MAR-74 09:08; Title: Author(s): Duane L. Stone/DLS;
Distribution: /DLS; Sub-Collections: RADC; Clerk: DLS;
Origin: <MCNAMARA>NIH.NLS;1, 14-DEC-73 06:30 JLM ;

Response to (hjournal,22353)

Reference: (hjournal,22353)

1

This note is to comment on the point of view expressed by Jake for the inclusion of her recommendations: 1, 2, 6, 8, and 9 in (using,udef3,7).

2

With respect to the developement of an analysis system, I would like to offer the following thoughts:

3

It seems that the major issues here are overhead requirement, cost to develop, and potential invasion of privacy. These may be traded-off against the hard to achieve benefit of understanding resource utilization better so that in turn better design decisions may be made in the future.

3a

Having a group devote some time the a generic design of what such a system might look like appears to be a reasonable idea because it would allow:

3b

Understanding the major functions required, 3b1

Estimating what operational overheads might be involved, 3b2

Exploration of different design alternatives, 3b3

Understanding how such a system would have to involve the user for differing levels of data collection, 3b4

[and most importantly] provide a uniform target around which a more detailed exploration could be predicated. 3b5

I would recommend that a subgroup of USING be consituted to come up with a generic design for such a system for the consideration of the membership. I believe that such an exercise would be both worthwhile and allow us to focus on what is really involved.

3c

With respect to the definition of a use policy or goal for the ARPANET, I agree that it would be nice if possible. However, to press for such a recommendation at this time might be difficult because of the volatile and changing nature of the network. Perhaps some guidelines could be established to serve as an evolving understanding of what the ARPANET is becomming. It seems that a number of coherent

Response to (hjournal,22353)

subnets are forming within the ARPANET, for example: ARPA Funded R&D groups, the Air Force Systems Command, the Army Material Command, etc....The degree of interaction between these apparent subnets needs to be better understood. Communities, such as the weather community, or the Seismic community, have potential for being very significant users. Maybe some understanding of their ARPANET use policy, goals, and plans might be a good starting point.

4

Isn't recommendation number 6, interfaceing other networks, somewhat removed from the purpose of the committee. I share the sentiment expressed, but think it is not appropriate to the report.

5

Although I applaud the sentiment of recommendation 8 and support it, isn't it also inappropriate to the committee's charter?

6

I believe that recommendation number 9, although a good observation, is not properly the pervue of the committee.

7

I tender these comments in response to receipt of the referenced Journal item because of a conviction that feedback and communications are the cornerstones of collaborative endeavors.

8

JI 12-MAR-74 13:37 30215

Response to (hjournal,22353)

(J30215) 12-MAR-74 13:37; Title: Author(s): Jean Iseli/JI;
Distribution: /USING; Keywords: user definition response reciprocity;
Sub-Collections: USING; Clerk: JI;
Origin: <MITRE-TIP>JAKE.NLS;1, 12-MAR-74 13:28 JI ;

WJC 12-MAR-74 09:22 30216

PURDUE problems with UCLA-CCN TSO LOGON

We have anaccount at CCN (ARP401, id:PAW, pass:MICLOG).
By the time you get this we may have found a way around
it but at present it is impossible for us to LOGON to TSO because
ve inadvertently CHANGEed our logon PROC to a non-legal
invalid procedure (whenever we attempt to
LOGON we get the following message:
LOGON FAILED
'LOGON' WAS NOT FOUND IN COMMAND BUFFER

Can you somhow help us out of this

Much thanks,

Bill Croft PURDUE@OFFICE-1

WJC 12-MAR-74 09:22 30216

PURDUE problems with UCLA-CCN TSO LOGON

(J30216) 12-MAR-74 09:22; Title: Author(s): William James Croft/WJC; Distribution: /BN WJC; Sub-Collections: ; Clerk: WJC;

Vacation carry-over

To Whom it May Concern:

I would like to request that one week of Philip Feldman's 1973 vacation be carried over to 1974. It will be difficult for Phil to take this remaining week before May 1, 1974 as the following facts will demonstrate:

1)Phil was on a week-long course the week of Feb. 3.

2) A week-long course scheduled for the week of Feb. 25 was cancelled and may be re-scheduled in the near future.

3)Phil was on a ten day business trip finishing March 8.

4)Phil will be going to Frobisher Bay on business on March 15 and will be there for one or two weeks.

5)The first working paper coming out of a contract Phil is supervising is due on May 1.

6)Because of these major and other minor trips made necessary by involvement in several project areas, it would be very inconvenient for Phil to take one week's vacation over the next few months.

For further details, please contact Philip Feldman, Supervisor-Business Planning at 870-5917.

PF 12-MAR-74 15:14 30218

Vacation carry-over

(JJ0218) 12-MAR-74 15:14; Title: Author(s): Phil Feldman/PF;
Distribution: /LHD; Sub-Collections: NIC; Clerk: PF;

Chapter 6 1

FORMATTING 1a

6.1 Introduction 1a1

JOVIAL formatting provides the capability of translating between a character buffer and data elements. The character buffer is a †character:formula or †character:variable and contains a string of any legal ASCII characters including the control characters. A data element is a †variable or †formula depending upon the direction of translation or it may be an †indexed:variable:range (see Section 10.4.4). 1a1a

.1 Formatting can be either list-directed or format-directed. When list-directed, the data elements determine the manner in which the character string is generated or scanned. When format-directed, an explicit †format:list controls scanning or generation of the character string. List-directed formatting provides a free-field "input" capability. 1a1a1

.2 Translation is accomplished by an †assignment:statement and formatting routines. The direction of translation is determined by the right and left side of the †assignment:statement. 1a1a2

.3 Translation from the character buffer into the data elements is accomplished by an †assignment:statement (Section 5.5) of the form: 1a1a3

†variable 1a1a3a

†indexed:variable:range ←=
†format:function:call ←; 1a1a3b

.4 ←FORMAT signals an †intrinsic:function:call to a formatting routine designed to translate between character buffers and data elements. Its first †parameter, †character:formula, provides the character buffer from which values are obtained, translated as necessary, and assigned to the data elements on the left of the †assignment:statement. 1a1a4

.5 If the optional second †parameter to ←FORMAT, †format:list, is present, it implies †format-directed formatting; but its place in the †parameter list is marked by a †comma for list-directed formatting if the third †parameter is given. †Format:list, defined in

Section 6.14 controls the scanning of the character buffer. 1a1a5

.6 The optional third ↓FORMAT †parameter, †procedure:name, names a contingency processing procedure to be called by the formatting routine. The details of the interface between the contingency procedure specified and the formatting routines are left to the implementer. The intent of the procedure is to process errors, end-of-line conditions, etc., without the necessity of returning from the formatting routine. For contingency action, the formatting routine will call the named procedure which might, for example, cause a line to be written or read and then return to the formatting routine to continue translation. 1a1a6

.7 Translation from data elements into a character buffer is accomplished by means of an †assignment:statement of the form: 1a1a7

†format:variable ↓= †formula ↓; 1a1a7a

†indexed:variable:range 1a1a7b

.8 Here ↓FORMAT signals a call to an intrinsic formatting routine defined to translate between data elements and character buffers. Its first †parameter, †character:variable, names the character buffer into which the translated values of the data elements given on the right of the †assignment:statement are put. 1a1a8

.9 The second ↓FORMAT †parameter is as described in Section 6.1.5 above except that the †format:list, if present, controls the generation of the character string in the buffer. The third ↓FORMAT †parameter is as described in Section 6.1.6 1a1a9

6.2 List-Directed Formatting 1a2

List-directed formatting provides for translation from a input string to data elements or from data elements to an output string according to the syntax rules described below. Fields in the character buffer are separated by spaces. ("Input" and "output" refer to the usual uses these character strings, but this is not a requirement, since a core-to-core transfer may be all that is actually involved.) 1a2a

.1 When the translation is from a character buffer (input), the formatting routine scans the character

buffer, separating the fields and translating each field according to the type of the corresponding †variable to be assigned a value from the field. The formatting routine must recognize that there may be more data elements than there are fields, and possibly provide for new input using the contingency procedure. 1a2a1

.2 For a †character:variable, the field is delimited by spaces just as for other †variables if there are no spaces which are a part of the field. If the character field contains spaces, it must be †additionally delimited by a prime immediately preceding the first and a prime immediately following the last of the characters which constitute the field. The primes merely delimit the size of the field and are not considered a part of the field to be passed on to the †character:variable. Two consecutive primes would denote a field of zero characters. Obviously, it is impossible to include a prime in an input field delimited by primes. A leading prime with no trailing prime is illegal. For a field where the first nonblank character is not a prime, the field starts with that character and includes all characters following it up to the next space or blank, and so may include primes. Primes may delimit fields that do not contain either spaces or primes. A list-directed input field cannot contain both spaces and primes. 1a2a2

.3 The characters from the input field are assigned to the †character:variable using the rules of assignment; i.e., they are left justified. If the input field is too long, excess bytes on the right are truncated before the assignment. If there are too few bytes in the input field, blanks are added at the right to match the size of the †variable before assignment. 1a2a3

.4 For †numeric:variables, the characters in the input field are syntactically analyzed to determine their form (integer, fixed, or floating type), converted appropriately as if they were JOVIAL †constants to the type of the †numeric:variable, and then transferred according to the rules of assignment. The forms that may occur in the input string do not permit all forms for JOVIAL †numeric:constants. In particular, they must fit the following syntactic form and, as said before, no spaces are permitted: 1a2a4

1a2a4a

←. digit 1a2a4b

.5 List-directed input to a †bit:variable is undefined. 1a2a5

.6 For list-directed output to a character buffer, the †formulas that comprise the data elements are evaluated and then translated to the appropriate character representation and placed in the character buffer. Any pair of fields is separated by a space. If the value of a †formula is zero, one zero is output. The formatting routine must recognize when the character buffer will not hold another field, and possibly provide for output and clearing of the buffer using the contingency procedure. 1a2a6

.7 Converson for list-directed output takes place according to the †formula types as follows: 1a2a7

a. †Character:formula. Spaces to the right of the last nonblank character are not included in the output, but would be restored if subsequently input to the same †variable. If the †formula contains spaces other than on the right, it is surrounded by primes as well as followed by a space on transfer to the character buffer. If the †formula contains primes, and the value is to be surrounded by primes, it should be noted that the result will not be legal input to the list-directed formatting routines. 1a2a7a

b. †Numeric:formula of floating type. The character representation of the floating number is a normalized, signed, fractional significand with a base 10 exrad. Trailing zeros are suppressed and the number of fractional positions is equal to (significant fraction bits/3.32)+1. The base 10 exrad is preceded by its sign and the letter ←E. The syntactic form for list-directed floating output is: 1a2a7b

←+ ←. digit ←E ←+ digit 1a2a7b1

c. †Numeric:formula of fixed or integer type. Fixed values with negative fraction bits or fraction bits greater than the number of significant bits are formatted as if they were †numeric:formulas of floating type (see b. above). For other fixed values and integer values, signed values are preceded with a plus or minus sign. The number of characters in the integer portion of the representation is equal to (integer bits /3.32)+1. The number of fractional digits is similarly computed. Integer and fractional

positions are separated by a decimal point. The syntactic form of the list-directed output is: 1a2a7c

```
.+     digit     .     digit                    1a2a7c1

                 .     digit                    1a2a7c2
```

d. †Bit:formula. List-directed output from †bit:formulas is undefined. 1a2a7d

.8 List-directed output is acceptable as list-directed input unless a †character:formula containing both spaces and primes is output. In general, it is not feasible to read list-directed output using format-directed input even though, for any given †constant found in a list-directed output field, there is a †format which can describe it. 1a2a8

.9 Given below are examples of a †character:variable of five bytes output by list-directed formatting and then input from the output buffer. The input and output values are shown as JOVIAL †character:constants and are consequently delimited by †primes. The internal †primes are indicated by the three †character code .$27. Note that the last input value is undefined because the output value contained a leading (embedded) blank and a prime. .b represents a blank or space. The leading field-separating blank is not shown; the trailing one is: 1a2a9

Output Value Input Value	Output Buffer	
'ABCbb' 'ABCbb'	ABCb	1a2a9a 1a2a9b
'bABCD' 'bABCD'	'bABCD'b	1a2a9c
'A$27Bbb' 'A$27Bbb'	A'Bb	1a2a9d
'AbBbb' 'AbBbb'	'AbB'b	1a2a9e
'bA$27Bb' undefined	'bA'B'b	1a2a9f

6.3 Format-Directed Formatting 1a3

With format-directed formatting, the picture of each character in the buffer is determined by its own †format. Except when †null:format is specified, each space and character is mentioned in the †format:list. 1a3a

.1 Basically, a †format tells how to interpret or generate a string of characters. A †format describes the data in a buffer, which may come from or go to an external medium; it is much like an †item:description as if applied to a †constant. The character strings are rather evanescent from the standpoint of the program, however; so the †formats must be applied to them dynamically--during actual execution of the program. 1a3a1

.2 Usually a string of characters is matched, through a †format, to a data element. However, there are some special †formats; the †insert:format describes character strings without reference to a data element, and the †skip:format causes a data element to be passed without matching it to a charter string. 1a3a2

6.4 †Insert:Format 1a4

†Insert:formats do not correspond to any data element. They may at times stand alone, particularly the form staring with ←/ and so be separated from other †formats by †commas. However, they are most likely to be used as a part of another †format and in these cases are not separated by †commas. (†Insert:formats are not permitted as part of a †null:format or †skip:format.) 1a4a

.1 In general, a †format specifies a number of character positions filled in ways dependent on the value of the corresponding data element. Let us call these the "effective" character positions. An †insert:format specifies the content of other character positions inserted preceding, among, or following the effective character positions for a data element. The positions, but not the contents, of effective character positions are affected by †insert:formats. The contents of effective character positions are the same regardless of the presence or absence of †insert:formats. Obvious uses of such †insert:formats are for spacing on a printed page between values corresponding to data elements, for inserting commas between certain digits in a long number, or for titular information. There is no output of spaces not actually coded using an †insert:format, or as part of a character output value, or as a separator for †null:formats. 1a4a1

.2 The effect of †count is as if the following ←S, ←/ code, or †character string in †quotation:marks had occurred that many times. A †count of ←1 is assumed if no †count is given. 1a4a2

.3 The use of ←S is a shorthand way of indicating spaces, most often between data elements but possibly separating effective character positions. ←S is equivalent to ←" ". The following five examples of †insert:formats are all equivalent and each would cause seven spaces to be placed in the output buffer or seven character positions to be ignored in the input buffer: 1a4a3

←SSSSSSS 1a4a3a

←SSS SSSS 1a4a3b

←4S2SS 1a4a3c

←5S 2S 1a4a3d

←7S 1a4a3e

.4 With †slash there is a difference. As an †insert:format, †slash followed by a †letter or †numeral is a system-dependent indicator of line, page, or other device control function. These may cause different results upon input and output. For example, the same code that causes the current record to be written upon output probably causes the next record to be read upon input. 1a4a4

.5 The †insert:format of a †character string in †quotation:marks is intended to permit any string of ASCII characters to be inserted as desired among effective character positions. It would seem preferable to use a †character:constant for the purpose, but †quotation:marks are used for delimiters instead of †primes because of an †insert:format is a part of a †format:list which is itself a †character:formula and consequently could be a †character:constant delimited by †primes. So the use of the †quotation:marks as delimiters avoids the problem of having a †character:constant occur within a †character:constant. Before †format analysis takes place, any three-†character codes and special two-†character codes occurring in a †character:constant that will become part of the †format:list are replaced with the single †characters they represent. Additional analysis of †insert:formats

using †quotation:marks as delimiters is necessary in the formatting routines to permit the occurrence of a †quotation:mark as an insert †character. A single †quotation:mark as an insert †character is represented by two †quotation:marks in succession. In general, internal strings of †quotation:marks within the delimiting †quotation:marks must be halved to get the effective insert †characters. The †insert:format ←"""" would therefore cause one quotation mark to be inserted (or one character position to be ignored on input). 1a4a5

.6 For input, †insert:formats delimited by †quotation:marks indicate positions in the character buffer to ignore. (There are no positions corresponding to the delimiting †quotation:marks or to half of the internal strings of two †quotation:marks.) 1a4a6

.7 For output, †insert:formats delimited by †quotation:marks indicate positions in the character buffer where the delimited character values are to be placed (the delimiting †quotation:marks and half of any internal strings of †quotation:marks are removed). 1a4a7

.8 The following three examples of †insert:formats are equivalent to each other: 1a4a8

←"/*/*/*/*" 1a4a8a

←"/*""/*""/*""/*" 1a4a8b

←4"/*" 1a4a8c

.9 The examples below are equivalent to each other but different from those just above: 1a4a9

←"/*/*/*/*" 1a4a9a

←4"/*" 1a4a9b

.10 Examples of the use of †insert:formats in connection with other †formats are given in the following sections. 1a4a10

6.5 †Skip:Format 1a5

A †skip:format causes the data element corresponding to this †format to be skipped on input or output. There is no corresponding field in the character buffer. 1a5a

6.6 †Character:Format 1a6

INSERT BOX 1a6a

.1 The †character:format is, in essence, a string of ←C's. A †count of ←1 is assumed if no †count is given. The total †counts of ←C's is the number of effective character positions in the buffer. (Effective character positions are those specified by †formats but not including †insert:format characters.) 1a6a1

.2 For output, the character value of the output data element is placed in the effective character positions of the buffer. If the data element has more bytes than there are ←C's, excess bytes on the right of the data element are truncated. If the data element has too few bytes, the field is padded with extra blanks on the right. For input, excess bytes from the field are truncated on the right or, if the field is sort, blanks are added on the right to match the size of the data element before assignment. 1a6a2

.3 An example of use of a †character:format is given in Section 6.7.3. 1a6a3

6.7 †Null:Format 1a7

A †null:format is permitted. It is indicated by means of an extra †comma in the †format:list. The †null:format consists of nothing but possibly a string of †spaces. 1a7a

.1 The †null:format indicates that conversion is the same as for list-directed formatting. For conversion purposes the input field starts with the first nonblank character at or after the current character of the buffer and ends with the first space after a nonblank character unless the first nonblank character is a prime; in this case the field starts with the character following the prime and terminates with the character that precedes the next prime. The current character for the next field will be the character following the space following the terminating prime. 1a7a1

.2 Output involving a †null:format is done in accordance with Section 6.2.6. 1a7a2

.3 The following table gives examples of two input buffer strings broken into fields in accordance with two †format:lists, the first consisting of two †null:formats followed by a †character:format and the second consisting of a †character:format followed by two †null:formats.

Since the description of the corresponding data element affects the meaning of the field, the ↑variable corresponding to each ↑null:format is assumed to be of character type and six bytes long. The ↑variable corresponding to the ↑character:formats is ten bytes long. The table shows the value of each field, as a JOVIAL ↑constant, determined from the ↑formats above and the input buffer strings on the left. (Note: ←b represents a blank and $27 is the three-↑character code for a prime within a JOVIAL ↑constant.) Blanks have been added or characters truncated to give ↑constants of the length corresponding to the ↑variable. 1a7a3

6.8 ↑Pattern:Format 1a8

INSERT BOX 1a8a

.1 The ↑numeral preceding ←B applies to the whole ↑pattern:format and indicates the "order", the number of bits of the data element to be associated with each effective character position. The optional ↑count preceding ←P simply is shorthand for indicating so many ←P's. A ↑count of ←1 is assumed if no ↑count is given. The total number of ←P's gives the number of effective character positions in the buffer. The bit groups are associated with characters in accordance with the table of ↑pattern:digits (Section 2.8.8). For each bit group in the table under "pattern" (as modified by "order") the letter or numeral under ↑pattern:digit is indicated for the corresponding effective character position in the buffer. 1a8a1

.2 Effective character positions are matched with bit groups starting at the right. On output, excessive bits at the left of the data element are discarded. If there are too few bits in the data element, zeros are added at the left before conversion to ↑pattern:digits. On input, the effective character positions of the field are treated as ↑pattern:digits, translated to bits, then assigned to the data element. Leading and trailing blanks are all treated as leading zeros and consequently do not affect the assigned value. Embedded blanks in the input field corresponding to the ↑pattern:format and any other characters not indicated as ↑pattern:digits for the indicated order are undefined. Blanks and other characters corresponding to ↑insert:formats are, of course, ignored on input. 1a8a2

.3 In the following table, the heads of the rows are

examples of character data element values whose bit patterns are shown in Section 6.8.4 below and the heads of the columns are †pattern:formats. ←S's occurring in the †formats are †insert:formats. It is assumed that character values are stored in eight-bit bytes in which the first four bits correspond to the head of the row in the table of characters (Figure 2-1). The body of the table below shows the contents of the output field corresponding to the character values and the †pattern:formats. 1a8a3

	←4B3PS3PS3PS3P	←5B5PS5P	1a8a3a
←Bah	←000 042 616 821	←00011 62Q11	1a8a3b
←Humbug	←487 56D 627 567	←28ELM M4TB7	1a8a3c

.4 The above table is based on the following bit configurations, from the table of ASCII characters, ticked off in 4 bit groups by primes and 5 bit groups by commas: 1a8a4

←Bah
01,00'001,0'0110,'0001'0,110'10,00'001,0'0001 1a8a4a

←Humbug
010,0'1000,'0111'0,101'01,10'110,1'0110,'0010'0,111'01
,01'011,0'0111 1a8a4b

.5 The table below gives examples of the values of JOVIAL †pattern:constants associated with fields of two input character strings in accordance with the list of †pattern:formats given at the top. These †constants could be assigned to †variables of any type. The ←S after the first comma is an †insert:format and causes one character of the input buffer to be skipped. 1a8a5

Input Buffer	Field	←4B5P,S1BP,3BPPP	1a8a5a
←bbAAbb1762	←1	←4B'AA'	
	←2	←1B'1'	
	←3	←3B'762'	1a8a5b
←4A3CEb217b	←1	←4B'4A3CE'	
	←2	undefined because	
the †pattern:digit ←2 does not occur for order ←1			
	←3	←3B'17'	1a8a5c

6.9 †Numeric:Formats 1a9

†Numeric:formats provide the specifications needed to convert from internal numeric value representations to external character strings and vice versa. 1a9a

.1 It is not required that a data element and its corresponding †format have the same type. 1a9a1

.2 On output, the value of the data element is first converted to a decimal string of the same type (integer to integer, fixed to fixed, floating to floating). A decimal fixed value looks like a floating JOVIAL †constant without scale factors. Consideration of the †format can avoid generating unnecessary digits. If the †format calls for a different numeric type, the decimal string is then converted accordingly. A †generalized:numeric:format does not cause decimal-to-decimal conversion. If the †format calls for rounding, it is decimal rounding performed on the final converted decimal string. 1a9a2

.3 On input, the input field is converted first to its internal representation, then to the type of the data element. If an input string does not match the †format, the translation is undefined. The input string which is left after all character positions corresonding to †insert:formats are deleted must represent a legal JOVIAL †numeric:constant; there can be no embedded blanks in it. If the †format is a †generalized:numeric:format, the input characters may be integer, floating without an ←E (†fixed:format), or floating with an ←E, and all decimal points must be explicit. In general, values and †formats must be such as to prevent the loss of significant signs and most-significant digits, or results will be erroneous and generally unpredictable. 1a9a3

.4 Following is the list of †format †signs that may appear in †numeric:formats, with an explanation of their meanings: 1a9a4

←N A character position that may contain any character that is part of a legal numeric field. 1a9a4a

←D An effective character position that will always contain a digit on output. On output it may contain a digit, a space, or a sign (←+ or ←−). 1a9a4b

←Z An effective character position that may contain a digit or a space on output. On input, it may contain a digit, a space or a sign (←+ or ←−). 1a9a4c

↓R Indicates that the digit string (significand in †floating:format) is to be rounded. On output, a decimal rounding is performed on the final converted decimal string. On input, a binary rounding is performed before the assignment to the variable. If ↓R is absent the digit string is truncated to the required length without rounding. No space is allocated for ↓R in the buffer. 1a9a4d

↓+ An effective character position for ↓+ or ↓- or space. 1a9a4e

↓- An effective character position for ↓- or space. 1a9a4f

↓. An effective character position for the decimal point, or a space. 1a9a4g

↓* The position of the understood decimal point. No space is allocated for ↓* in the buffer. 1a9a4h

.5 No plus or minus sign is printed (unless specified by an †insert:format) in any numeric field containing a zero value. 1a9a5

.6 †Numeric:formats provide for print suppression (replacement with blanks) under certain conditions. Character suppression except for ↓+ or ↓- is tied to ↓Z. Nothing, other than ↓+ or ↓- is suppressed in †formats not containing ↓Z. Any or all nonblank characters in †formats containing ↓Z's including insert characters may be suppressed depending on the actual suppression occurring in the effective character positions corresponding to ↓Z's. 1a9a6

.7 If an †insert:format immediately follows a ↓Z which is actually suppressed, the inserted characters are also suppressed. 1a9a7

.8 The decimal point corresponding to ↓. in a †format is suppressed if the nearest effective character position on the right (in the same field) corresponds to a ↓Z and is actually suppressed. 1a9a8

.9 Plus and minus signs are movable in a suppress context. If leading zeros corresponding to ↓Z's in the †numeric:format are suppressed on output, then the plus or minus corresponding to ↓+ or ↓- the the left of ↓Z's in the †format, and any †insert:format between the ↓+ or ↓- and the ↓Z's, are moved to the right the number of

spaces corresponding to the number of leading zeros suppressed. In the example below we have a table showing an †integer:format (with four †insert:formats as part of it) at the top, and several output values on the left. The output fields corresponding to the †format and the values are given on the right. (Subsequent input of these output fields using the same †format would cause the inserted characters to be ignored and give back the original values on the left.) 1a9a9

```
←"K = " +S3Z","3Z","3DS"MPH"                                    1a9a9a

←+523985612      ←K = + 523,985,612 MPH                         1a9a9b

←-500005610      ←K = - 500,005,610 MPH                         1a9a9c

←+000420000      ←K =       + 420,000 MPH                       1a9a9d

← 000000000      ←K =             000 MPH                       1a9a9e
```

6.10 †Generalized:Numeric:Format 1a10

(box) 1a10a

.1 The optional †count is a way of indicating that many ←N's. A †count of ←1 is assumed if no †count is given. On input, if the effective character positions corresponding to all the ←N's contain any legal integer or floating JOVIAL †constant (corresponding to †integer:, †floating:, or †fixed:formats) it will be accepted, converted in accordance with its self-evident type, rounded in binary if the ←R is present, and assigned to the †variable. 1a10a1

.2 The following table shows the values of JOVIAL †numeric:constants associated with fields of two input character strings in accordance with the list of †generalized:numeric:formats given at the top. The †minus:sign is not a part of the †constant but must be used in asssigning a value to the corresponding †variable. 1a10a2

.3 On output, conversion takes place as integer to †integer:format, fixed to †fixed:format, and floating to †floating:format. Minus signs are output; plus signs are not. Significands and fraction digits are limited by the field size. If there is not enough room for all integer digits on output, the field is illegal. Excess fraction

digits are truncated or rounded depending upon the absence or presence of the ↓R. Zeros are suplied between the point and the significand digits for fixed values with less than zero fraction bits or more fraction bits than the size. 1a10a3

6.11 †Integer:Format 1a11

INSERT BOX 1a11a

.1 As before, a †count followed by a ↓D or ↓Z has the same meaning as that number of ↓D's or [↓Z's, respectively. A †count of ↓1 is assumed if the optional †count is missing. †Insert:formats are permitted as described above. The †integer:format specifies an integer field of specific size in the buffer, for any numeric data element. For output, if the data element is not an integer, it will first be converted to a decimal representation, then to a decimal integer (rounding off any fraction if ↓R is in the †format, truncating the fraction otherwise). On input, the field is converted to internal integer form and then "assigned" to the data element (there can be no decimal point or fraction digits). If ↓R is in the †format, binary rounding occurs on assignment to †variables with missing low-order integer bits. 1a11a1

.2 The maximum number of decimal digits in the field is the total of the number of ↓D's and ↓Z's in the †format. If they are all ↓D's, all digits, even leading zeros, are printed on output or expected on input. If there are any ↓Z's in the †format, leading zeros are suppressed, but no more are blanked than there are ↓Z's. If the rightmost ↓D or ↓Z is a ↓Z, the non-blanked digits are left justified in the field defined by the ↓D's and ↓Z's. If the rightmost ↓D or ↓Z is a ↓D, the non-blanked digits are right justified and the sign, if there is one, is moved to the right by the number suppressed, leading zeros. 1a11a2

.3 ↓+ is the †format means print ↓+ if the value is positive, ↓- in the †format means print ↓- if the value is negative and space if the value is zero or positive. 1a11a3

.4 In the table below, the values at the left are used with each of the †integer:formats at the top to give the printed outputs in the body of the table: 1a11a4

.5 The following table illustrates the use of

†insert:formats with †integer:formats on input. The character strings in the input buffer are separated into fields in accordance with two †format:lists. The first has an †insert:format followed by an †integer:format and then another †insert:format. The †commas separating the †insert:formats from the †integer:format are not necessary, and only one field corresponding to a data element is described. The second †format:list consists of a †null:format followed by an †integer:format that has an †insert:format as part of it. Assuming that the †variable corresponding to the †null:format is of character type, the table shows the values of the input fields as JOVIAL †constants. 1a11a5

6.12 †Fixed:Format 1a12

INSERT BOX 1a12a

.1 A †count followed by ←D, ←Z, or ←* has the same meaning as that number of ←D's, Z's or ←*'s respectively. A †count of ←1 is assumed if the optional †count is missing. In the †fixed:format, the †decimal:point is the position of the actual printed decimal point and it occupies an effective character position in the field. A single †asterisk, however, is the position of an implied decimal point; the decimal point is not present in the character buffer, but the number is treated as if it were. If there is a sequence of ←n †asterisks, where ←n is ←2 or more, it is understood that ←n-1 trailing digits of the integer part of the buffer (beyond those specified by the †integer:part of the †format) are missing--or that ←n-1 leading digits of the fraction part of the buffer are missing. Zero suppression is not permitted to the right of ←*, the implied decimal point, in a †fixed:format. 1a12a1

.2 One effective character position corresponds to each ←Z, each ←D, a †plus:sign or †minus:sign, and a †decimal:point, but not to any †asterisks. The position of the decimal point, actual or implied, is fixed in (or outside) a field specified by a †fixed:format. It does not change position (although it may be suppressed if the value is entirely zero) with changes in value or suppression of leading or trailing zeros. 1a12a2

.3 On output, the suppression of leading zeros and closely associated †insert:formats and the moving of plus and minus signs is the same as for a right-justified †integer:format. Trailing zeros corresponding to ←Z's

following the explicit decimal point are suppressed. If the zero fraction digit immediately to the right of the explicit decimal point or immediately to the right of an †insert:format is suppressed, then the decimal point or †insert:format is also suppressed. 1a12a3

.4 Considering only the effective character positions, anything can be input using a †fixed:format that could be output using that same †fixed:format or any legal †fixed:format derived from that one by replacing some of the ←'Z's with ←'D's. 1a12a4

.5 The presence of ←R means the fixed decimal string is to be rounded on output to the decimal precision specified by the †format, or that the binary value is to be rounded on input before assignment to the †variable. ←+ or ←- means the same as in †integer:format. 1a12a5

.6 Output examples showing the use of each of the †fixed:formats at the top with each of the values on the left are given in the body of the table below: 1a12a6

.7 Subsequent input of the values shown in the above table in accordance with the †formats at the top would give the values shown below (note that precision may be lost due to rounding or truncation on input): 1a12a7

6.13 †Floating:Format 1a13

INSERT BOX 1a13a

.1 The †significand of a †floating:format is much like a †fixed:format, but it is more restricted. The only zeros that may be suppressed are trailing zeros to the right of an explicit decimal point. The ←+ or ←- have the same significance as in other †formats--they indicate what to do with the sign of the printed (output) †significand. †Count has the same meaning as in †fixed:format. 1a13a1

.2 The †exrad of a †floating:format is like an †integer:format with the added capability of the kind of †fixed:format that produces integer output. An exrad is always an integer and this †format allows it to be output without zero suppression or with zero suppression and justified either left or right. 1a13a2

.3 If the ←R is present, the significand of the value is rounded before input or output. The ←E is output or expected in the position indicated in the †format and

serves to identify the field as a floating value. The significand of the value precedes the ↓E as specified by the †significand of the †format. The exrad of the value follows the ↓E, again, as specified by the †exrad of the †format. 1a13a3

.4 If an output significand is zero, the corresponding exrad is also zero. If a zero significand is completely suppressed, so is an explicit decimal point. If a zero exrad is completely suppressed, so is the ↓E. If both the significand and the exrad are completely suppressed, so are all the nonblank characters in the field. 1a13a4

.5 Let ↓v be the value of a number output in accordance with a †floating:format. Then 1a13a5

↓v = m x 10 1a13a5a

where ↓m is the value of the output significand and ↓c is the value of the output exrad (including the mathematical sign in both cases). If the significand is not completely zero, the leftmost digit in the field and is not zero. With a given †format, every non-zero significand has an absolute value ↓M such that 1a13a6

↓10 < M < 10 1a13a6a

where ↓n is an integer determined entirely from consideration of the †format. The value of ↓n is zero or a positive integer equal to the number of ↓D's and †asterisks to the left of the †decimal:point or the rightmost †asterisk in the expanded †significand of the †floating:format--unless there are no ↓D's to the left of the †asterisk or †asterisks. If all the ↓D's are to the right of one or more †asterisks in the expanded †significand, then ↓n is negative (or zero) and its value is none minus the number of †asterisks. Thus, non-zero ↓M cannot be less than one tenth unless the †significand has an implied decimal point such that there are one or more missing digits on the left of the output significand. 1a13a7

.6 In the following example, two values are shown as they would be output in accordance with several †floating:formats. 1a13a8

6.14 †Format:List 1a14

INSERT BOX 1a14a

.1 Note that two definitions are given for a †format:list. First, a †format:list is a †character:formula with a value that is a string of †characters. This string of †characters must be analyzed by the formatting routines and recognized as a †format:list fitting the second definition. If the †character:formula is a †character:constant, it would be highly desirable to have the analysis done at compile time; if the †character:formula is not a †character:constant, the analysis must be done at execution time and increases the program's run time. 1a14a1

.2 The †format:list in †parentheses and preceded by a †count has the same significance as the indicated number of iterations of the enclosed list, separated by †commas. The number of †commas is highly significant with regard to indications of †null:formats. The expanded †format:list has †commas inserted between iterations, but not at either end. Example: 1a14a2

```
←2(2(2S3Z "," 3D), 10C                                    1a14a2a
```

.3 The above †format:list means the same as the expanded †format:list: 1a14a3

```
←2(2S3Z "," 3D), 10C, 2(2S3Z "," 3D), 10C                 1a14a3a
```

which means the same as the completely expanded †format:list of six †formats: 1a14a4

```
←2S3Z "," 3D, 2S3Z "," 3D, 10C, 2S3Z "," 3D, 2S3Z "," 3D, 10C
                                                          1a14a4a
```

.4 After expansion of parenthesized †format:lists, a †format:list is a list of †formats separated by †commas. An †insert:format may, without †commas, be included in other †formats except †null:formats and †skip:formats. †Spaces in a †format:list, if not part of an †insert:format, are ignored. Since †null:formats are permitted, a legal †format:list may begin or end with a †comma or have †commas together with no explicit †format between them. For example: 1a14a5

```
←,3S,,X15C,4B10P,,                                        1a14a5a
```

is a †format:list of eight †formats of the following respective types: 1a14a6

†null:format 1a14a6a

```
    !insert:format of three spaces                                  1a14a6b

!null:format                                                        1a14a7

    !skip:format                                                    1a14a7a

    !character:format of 15 characters                              1a14a7b

    !pattern:format of ten hexadecimal characters                   1a14a7c

    !null:format                                                    1a14a7d

    !null:format                                                    1a14a7e
```

.5 The !format:list: 1a14a8

```
    ←,3S,,X,15C,4BP "," 3P "," 3P "," 3P,,                         1a14a8a
```

is the same as the above with the !insert:format "," added to the !pattern:format for separating the digits with commas. These eight !formats correspond to seven data elements. The !insert:format ←3S serves only to indicate three additional spaces between the first two data elements which had !null:formats. 1a14a9

.6 The rules do not permit omitting !commas between parenthesized parts of a !format:list. 1a14a10

←2(7D)3(6D) is not a !format:list. 1a14a10a

←2(7D) means the same as ←7D,7D (two !formats). 1a14a10b

←2(7D,) means the same as ←7D,,7D, (four !formats). 1a14a10c

.7 Upon execution of a call to the ←FORMAT routine, the individual !formats of the expanded !format:list (including !null:formats and !skip:formats, but not stand alone !insert:formats) are matched one by one, starting from the left, with the individual data elements. If there are more !formats than are needed, the extra !formats at the right of the expanded !format:list are not used. Even stand alone !insert:formats immediately following the !format matched with the last data element are left unused. If there are not enough !formats to match all the data elements, the entire !format:list is repeated as many times as necessary--it is as if the entire !format:list were enclosed in !parentheses and a

sufficiently high †count prefixed to the parenthesized list. 1a14a11

.8 If data elements are output using an appropriate †format:list and then the same data elements are input using the same †format:list, the input values should be identical to the original output values except where precision might be lost due to the truncation or rounding of values. 1a14a12

6.15 Input and Output 1a15

Input and output †primitives are not a part of the JOVIAL language. Aspects of input/output are quite system dependent and, therefore, including input/output in JOVIAL would mean picking some particular input/output scheme or designing one especially for JOVIAL. Instead, JOVIAL is designed s that perhaps with the use of system routines, JOVIAL programs (or systems) can be made to interface with many input/output schemes. JOVIAL even provides the capability to program input/output systems. 1a15a

.1 The following is a list of JOVIAL capabilities as they might be used for input/output purposes: 1a15a1

a. Reference external procedures for ←OPEN, ←CLOSE, ←READ, ←INSERT, etc. 1a15a1a

b. Make system calls for performing actual read/write operations. 1a15a1b

c. Pass †parameters by address (tables and data blocks) for transmitting and receiving records. 1a15a1c

d. Core to core conversion (←FORMAT). 1a15a1d

e. Pass †character:formulas; e.g., file name. 1a15a1e

f. Symbol generation (parameterized define) for generating parameter tables, file control blocks, etc. 1a15a1f

.2 Core-to-core conversion using ←FORMAT (see Section 6.1.3 and 6.1.7) is defined herein for data types of all kinds to and from character strings. This is useful for input/output involving character files but not binary files. Since core-to-core moving of bit strings is intrinsic to JOVIAL, nothing corresponding to ←FORMAT is needed for binary records (use data blocks or tables). 1a15a2

.3 Core-to-core conversion of numeric values between binary and decimal is carried out using standard algorithms. Conversion of integer values is exact if the capacities discussed below are not exceeded. Binary to decimal conversion applies only to formatting. The limit on accuracy is determined solely by the description of the output field. Enough extra digits are produced by the converson method to insure the accuracy implied by the field width and the rounding or truncating expected. A correction increment is added to the appropriate fraction digit before rounding or truncating to compensate for inherent losses. 1a15a3

.4 Decimal to binary conversion applies both to formatting and to the derivation of internal representation for †numeric:constants. Enough extra bits are developed to insure the accuracy implied by the size of the receiving registers, with the following limitation. No implementer is required to use more magnitude bits than 150% of the maximum size for the representation of numeric values. The 150% figure is the most required of the total of integer and fraction bits. The binary point may be anywhere within or to the left or right of these bits. 1a15a4

JOVIAL Manual--Chapter 6

Contains - 8 †, plus structure

DHC 12-MAR-74 18:33 30220

Gripe and Comments idnets foul up (?)

Somebody: What the blazes is going on here????

Some time back, I received a couple of test messages for idents NETGRIPE and NETCOMMENTS (or maybe it was NETIDEAS -- that'snot too important). I recently attempted to send journal mail to them and discovered they do not exist (anymore?) Jake calims no knowledge of what is going on, I think jean sent the test messages. I now receive a note which coincidentally mentions the formation of NGRP and NCMT for gripes and comments, respectively.

Will anyone who has some informattion about this mess please let the rest of us in on it? (mdk, dhc, njn, ji, jake, and anyone else you can think of). Dave.

1

DHC 12-MAR-74 18:33 30220

Gripe and Comments idnets foul up (?)

(J30220) 12-MAR-74 18:33; Title: Author(s): David H. Crocker/DHC;
Distribution: /MDK NJN JI JAKE; Sub-Collections: NIC; Clerk: DHC;

NJN 13-MAR-74 09:19 30221

Response to (hjournal,22353,)

Following are my replies to Jake's comments on the UDEF report. 1

USERS OF THE ARPANET 1a

I included the table of hosts because it was the only statistic we had, though I do not believe it is very useful in defining the User community. We are interested in people, not in resources. Knowing that universites and commercial establishments have bigger and better computing facilities than government institutions does not imply that there are more government people using the network than others. All it says is that they are forced to use other facilities than their own. If we could prove that there was a direct relationship between type of site and usage of the net, then the statistics would be valuable; now they are not. 1a1

The same principle applies to geographic distribution. It might be the case that 90% of the network users live in the MidWest and dial long-distance into computing facilities on the two coasts. We just can't prove anything about users from the known resources. The information about geographic distribution may be included in the report with the same caveat as the other statistics. 1a2

It is mentioned in the section on Access to the net that TIP and ANTS users have the most direct relation and usage of the net. Maybe it needs better wording, more explanation (a comment that applies to the report as a whole -- it is one of my stylistic inadequacies.) 1a3

PROFESSIONAL INTEREST 1b

I omitted professors and students because their function as users is not to be academics, but to do other things covered under other categories-- like programming, applications, committee work. One could alternatively list them under every category. 1b1

HOW MANY USERS 1c

It seemed obvious to me that slots are limited; or if they weren't now , then they would be soon. Several sites are already working on this problem. It seems to me to be a question of availability of resources rather than who the users are. I believe that the resource allocators will come up with as fair a mechanism as possible. 1c1

FUTURE USERS 1d

Are we to say who the future users will be or should be? The former would be difficult, because we can't even say who they are now. But it is something we can act on by setting up an analysis system, so in the future we will know. The latter question, of who they should be, is out of our range. That is a decision to be made by ARPA managers. If they want our opinion we had better have something to back up the "diversified network" theory. To you, to me, to most others we know, it is obvious that diversity is best. I would like to hear some argument s for both sides, and particularly for the other side, before forming an opinion to be passed on to ARPA. Just making the statemnt that diversity is best is worthless. 1d1

RECOMMENDATIONS 1e

I agree that the recommendations are a little weak, or vague. Better ideas and more detailed specs for analysing network users and usage would be appreciated. I simply suggested soem things that I thought could be done NOW. (By the way, I have gleaned useful information in the past from PI reports, at least about the nature of the research being done.) I suggested working WITH the Performance Measurement Lab in defining the specifics of how to obtain user information. This is not throwing away the job. PML must know how many people use particular services and what their type of usage is, before they can make any other analysis. Why do the job twice? It only aggravates those people providing the services. 1e1

As for recommendations 2, 6, 8, and 9 from UDEF3, 2 is covered in my comments on future users above, and the remainder are not appropriate to the issue of defining the user population, and in some cases, not appropriate to USING. 1e2

NJN 13-MAR-74 09:19 30221

Response to (hjournal,22353,)

(J30221) 13-MAR-74 09:19; Title: Author(s): Nancy J. Neigus/NJN;
Distribution: /UDEF; Sub-Collections: NIC UDEF; Clerk: NJN;

DHC 13-MAR-74 13:37 30222

Documents

Lynn -- I just looked at copies of DOC.LAR versions 1, 2, and 3, for TXTVERT, CONVERT and LTSET.

LTSET had line numbers and no lines between statements. Whoops. Other than that it looked ok.

TXTVERT and CONVET looked ok also.

Print a copy of each of them, on you terminal, and let their respective authors (kampe and urban) review them. If they have not complaints, go ahead and print them.

Dave.

Documents

(J30222) 13-MAR-74 13:37; Title: Author(s): David H. Crocker/DHC; Distribution: /LYNN; Sub-Collections: NIC; Clerk: DHC;

DVN 13-MAR-74 20:16 30223

Whereabouts of DSA Masters from January 23 and Febraury 6th

The DAS master file from January 23, <kerns,jan23-dsamaster.;1,> has been archive from your directory at Office 1. 1

T<kerns,feb6-dsamaster.;1,> is reposing in your directory waiting for archive. It will probably go to the archive tomorrow sometime and leave your page allocation. 2

In the process both files lived for a while in the directory <documentation> at ARC and could be retrieved off routine dumps around March 8. 3

DVN 13-MAR-74 20:16 30223

Whereabouts of DSA Masters from January 23 and Febraury 6th

(J30223) 13-MAR-74 20:16; Title: Author(s): Dirk H. Van Nouhuys/DVN; Distribution: /CGK EKM(fyi); Sub-Collections: DEIS; Clerk: DVN;

HI THERE, BOB. WE'RE ON THE NETWORK. HOW ABOUT LUNCH? 1

(J30226) 14-MAR-74 14:54; Title: Author(s): Janice Fain/JF4;
Distribution: /RAY2; Sub-Collections: NIC; Clerk: JF4;

JI 14-MAR-74 18:37 30228

Gripe and Comments idents [cleared up] response to (30220)

Dave: In response to your (30220,) - you are right. At the outset, we did establish the idents netcomment and netgripe, however, we discovered that in order to log in under an ident, that those two would simply not work because they consisted of to many characters. So, we abreviated them in order that we might log in under the abreviated idents to update the initial files that have been set up for them in the directory <using>. This should make no difference because the feedback program will do the mailing to the idents and the user will be prompted as to whether he is sending a comment or a gripe.

Since there was no other direct and easy way out, we abbreviated the idents as I am sure you will agree is a reasonable compromise. Now, when journal mail arrives to the initial files associated with the idents, it can be processed and the files updated easily.

If you can see a better solution, I would be pleased to consider it. Sorry for any confusion over the evolution of an interim solution.
Jean

1

JI 14-MAR-74 18:37 30228

Gripe and Comments idents [cleared up] response to (30220)

(J30228) 14-MAR-74 18:37; Title: Author(s): Jean Iseli/JI;
Distribution: /MDK NJN JAKE DHC(hope this clear it up..incidentally ,
read [22407,]) FEEDBACK; Keywords: feedback clarification;
Sub-Collections: FEEDBACK; Clerk: JI;

JI 14-MAR-74 18:44 30229

Response to(22407) , Crisis Resolved

susan: If possible, we would like to maintain feedback as the ident for the group until the group has served its purpose. As for the two abbreviated idents, they were necessitated [upon advice of the NLS folk] by the requirement to be able to log in as either to update and process the initial files maintained for them in <using>. Journal mail will be send to both initial files by a program to be called feedback [a prototype is being debugged] which prompts the use for relevent information [like is this a gripe or comment] and/or extracts the information from the attributes associated wwith the user directory.

Thank you for your interest in the feedback committee work. I think it is healthy that we are all working in the same direction and will make sure that all future work is addressed to your attention....Jean 1

JI 14-MAR-74 18:44 30229

Response to(22407) , Crisis Resolved

(J30229) 14-MAR-74 18:44; Title: Author(s): Jean Iseli/JI;
Distribution: /SRL JCN; Keywords: feedback commentary; Sub-Collections:
FEEDBACK; Clerk: JI;

MAW 15-MAR-74 07:49 30230

WWDMS MEETING

1. A meeting was held on 12 Mar 74 at 0830 to discuss the ramifications of the request for 2 persons to be part of a test-team for evaluating file maintenence of WWDMS. 1

2. We determined that we lacked complete information as to the nature of the evaluation effort at Gunter AFB on 25 Mar - 12 Apr. 2

3. A preliminary evaluation of personnel indicated that Lt. Wingfield, D. Bergstrom, Capt. Daughtry, R. Luizzi, and D. VanAlstine would be available for all or a portion of the three week period. Although no one has experience in WWDMS, each has experience in setting up general test procedures. 3

4. It was decided to remain aware of this upcoming effort and ascertain more complete information about RADC's involvement in this evaluation. 4

5. A call was made to Capt. Martin, WWDMS coordinator, for more information. He told us: 5

a. that Rome and Data Services would be responsible for providing the expertise and that the other services would be at Gunter to learn to read our reports. 5a

b. no one else knows much about WWDMS either. 5b

c. we will have a three day class on WWDMS and then proceed to develop DMS test scenarios. We will not code or run them. 5c

d. on April 22 Rome and Data Services will go to JTSA and run the tests. 5d

e. he (Capt. Martin) will try to obtain a fund citation to help defray the cost of travel. 5e

f. if anyone reads this please let me know. M. Wingfield. 5f

MAW 15-MAR-74 07:49 30230

WWDMS MEETING

(J30230) 15-MAR-74 07:49; Title: Author(s): Mike A. Wingfield/MAW;
Distribution: /RFI DFB DLD2 RAL; Sub-Collections: NIC; Clerk: MAW;
Origin: <WINGFIELD>MEETING.NLS;1, 12-MAR-74 11:46 MAW ;

MIKE 15-MAR-74 10:13 30231

I didn't know whether you'd received your answers from me or not.

Responses to Phil Weintraub's Questionnaire 1

Background 1a

1. Please describe, to the best of your ability, your working expertise. 1a1

minimal in most areas; What does this mean? 1a1a

In this group I'm fairly competent in the theory of designing Delphi questionnaires, but this is somewhat akin to excelling in the manufacture of buggy whips at the turn of the century. 1a1b

I think SPRITE is a valuable tool to the group; even if the study itself flops, we're still able to talk about sophisticated methodological tools and their developent in the group 1a1c

I guess you could put in that I am the "Wired City" freak in the group.....contacts in the invisible colleg of planners and researchers doing work in the area of future communications services. 1a1d

2. Briefly describe all of the projects you have worked on in this department. Also indicate the specialized areas or techniques used to effectively conduct these studies 1a2

Shit, this is a big question..... 1a2a

Analysis of difference between different groups of respondents to Delphi questionnaires (in-house and out-house) with an eye to improving the efficiency of the technique 1a2b

Preparation of the report "Computer Based Services of the Seventies" around Larry's mathamatical projections. 1a2c

pretty much an editorial function 1a2c1

Delphi Study: "The Future of Communications Services in the Home" 1a2d

I don't know what you can say about this.....It was unique in that it used two panels (methodological advance). It was designed to explore a few new areas (degree of improvement offered by wired city type services, e.g.) and also to confirm some of the stuff that ADL and SRI had done regarding apparent

I didn't know whether you'd received your answers from me or not.

attractiveness of services, as perceived by potential users. 1a2d1

development of a new technology assessment research methodology....SPRITE 1a2e

overview of Computer Augmented Management Systems (computer conferencing) and Bell's possible particpation in such (participating as users , and possibly as marketers or promoters) 1a2f

coordinating work of the BNR guys in Computer Mediated Interaction, or CMI (computer conferencing, again) 1a2g

"project management" if you will 1a2g1

managing the evolution of BPG into an augmented workshop 1a2h

obviously the wording of this is critical, since either DMA, LHD, MTB, or IMM could be said to be "managing" this. 1a2h1

I seem to be learning a lot about how the installers and repairmen work, although I don't think that is the sort of experience you had in mind 1a2h2

liason wth Sears in Toronto on their (Automated Order Service trial (using t.t phones for catalogue shopping from home 1a2i

important here to word it properly; can't imply that Bell Canada and Sears are in this together right from the start, because they went ahead without us (inspite of us, even) 1a2i1

Monitor of the Business Planning Chess Scoreboard 1a2j

Briefly describe the projects you are currently working on.Indicate all specialized techniques you will utilize to conduct the study. 1a3

Sprite Home Technology Assessment 1a3a

developing a presentation for the senior management on Computer Augmented Management Systems 1a3b

Considering your academic and working background, what types of studies do you see yourself as doing most effectively. 1a4

I didn't know whether you'd received your answers from me or not.

studies of sleeping, eating, and drinking from the home 1a4a

methodological development and analysis (a la Delphi or Sprite, or Cross Impact although that's getting into PF's area) 1a4b

analysis of behavioural factors affecting implementation of new technology 1a4c

presentation of fairly complex concepts to different audiences (not much of a calling for this I know, but I seem to be doin it a lot anyway.) 1a4d

critical reviews of various drinking establishments (international background especially valuable here) 1a4e

Have you worked for any other departments in this Company, before joining the Business Planning Group. 1b

I like this question ! 1b1

NO 1b2

If yes, please describe each of those jobs briefly, indicating any areas of expertise required for the job. 1b3

no no no no no no no no no no no no no no no no, I said. 1b3a

Please list all organizations and insitutions that you have contacted while you have been a member of the Business Planning Group.(Include all Think Tanks). 1c

You've got to be kidding ! 1c1

Alberta Teachers Association 1c1a

Applied Futures Incorporated (Bill Simmons' group) 1c1b

Association of Rectors and Principals of Quebec 1c1c

Avon Products; Canada 1c1d

Bell Northern Research 1c1e

Bowen-Mann Canada Ltd. 1c1f

CSF-Thomson; France 1c1g

Canadian Dep't. of Public Works 1c1h

I didn't know whether you'd received your answers from me or not.

Canadian Department of Communication	1c1i
Canadian Post Office	1c1j
Carlton University (School of Journalism)	1c1k
Contemporary Research Center	1c1l
Fujitsu Ltd.; Japan	1c1m
General Electric Co. Inc. (Business Environment Studies)	1c1n
Hudson Institute	1c1o
Institute for the Future	1c1p
MacMillan Bloedel Limited	1c1q
NASA Ames Research Center	1c1r
New Jersey Dep't. of Education	1c1s
Newark College of Engineering (Center for Technology Assessment)	1c1t
ORBA Information Ltd.	1c1u
Ontario Department of Transport and Communication	1c1v
Ontario Insttue for Studies in Education (OISE)	1c1w
Pulp and Paper Research Council of Canada	1c1x
Rome Air Development Center (Rome, New York SAC Base)	1c1y
Sir George Williams University	1c1z
Dep't. of Management	1c1z1
Dep't. of Sociology	1c1z2
Stanford Research Institute	1c1a@
The Futures Group	1c1aa
United States Dep't. of Commerce (Office of Telecommunications)	1c1ab
United States Post Office	1c1ac

MIKE 15-MAR-74 10:13 30231

I didn't know whether you'd received your answers from me or not.

Washington University (Program in Technology and Human Affairs) 1c1ad

Please list all of the studies that you have conducted while being a member of the BPG (list titles of reports, papers, etc.) 1d

there are just soooooooo many; see (bedford,biblio,) for the BPG publication list) 1d1

If you had to convince the Executive that the BPG should spin off into a separate group, what type of an argument would you put forth to convince them that it is the best possible thing for the Company to do. 1e

I would tell them that Phil Weintraub is 100% behind this project, and that if his recommendations aren't carried out exactly as he wishes, he will resign immediately.....That should do it ! 1e1

Include a personal resume outlining educational and work experience. 1f

Personal History 1f1

Michael T. Bedford was born in Oakland, California on February 2, 1946. He has lived in Montreal since 1961. Married in 1968, he lives with his wife Jean on a farm about forty miles from Monrtreal. 1f1a

Education 1f2

Mr. Bedford received his secondary education at Westmount High School before entering Sir George Williams University. He graduated from S.G.W.U. (B.Com.) on the Dean's List in 1970. In 1972 he obtained his Master of Business Administration degree from McGill University. In both his undergraduate and graduate studies, he concentrated on the Marketing and Market Research disciplines. His Master's thesis was based on a study he conducted for Bell Canada; the study involved the application of the Delphi technique (a technological forecasting methodology) to the problem of estimating the demand for some possible new communications services. 1f2a

Work Experience = NIL, except for that outlined above. 1f3

MIKE 15-MAR-74 10:13 30231

I didn't know whether you'd received your answers from me or not.

(J30231) 15-MAR-74 10:13; Title: Author(s): Michael T. Bedford/MIKE; Distribution: /PIW; Keywords: Playboy Club; Sub-Collections: NIC; Clerk: MIKE;

MAP 15-MAR-74 12:30 30232

a small fix

hi--

in response to an observation of chris thomas, I have attempted to re-clarify the description of the verify request (making it say that it really affects change, locate, and next, but not print; and pointing out that it's mainly good for not being drowned in output from multi-line changes). won't bother to re-xfer the whole schmear, but thought I'd save everybody else the trouble of pointing it out to me.

cheers, map

p.s. to dave grothe: consider yourself nagged at (everybody else has confirmed receipt of my last week's biggy)

MAP 15-MAR-74 12:30 30232

a small fix

(J30232) 15-MAR-74 12:30; Title: Author(s): Michael A. Padlipsky/MAP; Distribution: /NETED; Sub-Collections: NIC NETED; Clerk: MAP;

RJC 15-MAR-74 12:54 30233

Tickler for the Week of 18 March 1974

(mm4) 18 March - Monday 1

Col Thayer will be here as of late afternoon permanently (?) so we start back with his Signature Block 1a

John McNamara is Acting Branch Chief this week. 1b

Sam DiNitto is Acting Chief for Section Chief 1c

0830 hrs. Branch Chief's Meeting 1d

WWMCCS ADP Requirements inputs on paper to be reviewed & commented on by all participants(cy ea member)from AFDSDC Gunter AFB 1e

On or about TODAY - Mr. Robert Majors (AFDSDC) will visit RADC/ISI in regards to WWDMS and RADC/ISI commitment 1f

F. Tomaini & R. Nelson - TDY 1g

Presentation to be held on "Advanced Techniques in Structured Programming" given by James Culp - RCA - Bldg. 3 - Conference Room 1a at 1400 hrs. 1h

(mt4) 19 March - Tuesday 2

F. Tomaini & R. Nelson - TDY 2a

Collect topic write-ups today by noon for confessions. 2b

(mw4) 20 March - Wednesday 3

Due Date - ISI/ISIS/ISIM - Forward Nominations to ISM - Remedial Education Courses (For GS-7 and below) 3a

F. Tomaini & R. Nelson - TDY 3b

ISI Confessions 0830 hrs. 3c

R & T Selection of the Month is due in ISM. 3d

(mth4) 21 March - Thursday 4

F. Tomaini & R. Nelson - TDY 4a

Commander's Supervisors Call - 10:00 hrs. - Bldg 106 - Auditorium 4b

General Electric IR&D Review of Proposed FY-74 Program - Bldg. 106, Room A119 - 0830 hrs. 4c

Tickler for the Week of 18 March 1974

0830 hrs. Branch Chief's Meeting 4d

Laboratory Activity Reports due today: Bucciero must have them by 1000, ISM must have them by 1100, and DOT must have them by 1600. 4e

(mf4) 22 March - Friday 5

F. Tomaini & R. Nelson - TDY 5a

Bobbie: Travel figures due by noon. 5b

Due Date - Course on "Effective Presentation Sumbit Names only to ISM NLT 22 March 5c

RJC 15-MAR-74 12:54 30233

Tickler for the Week of 18 March 1974

(J30233) 15-MAR-74 12:54; Title: Author(s): Roberta J. Carrier/RJC;
Distribution: /RADC; Sub-Collections: NIC RADC; Clerk: RJC;

JBP 15-MAR-74 18:12 30234

lynn:
thanks for the recent network measurement notes, any luck finding copies of the older ones i am missing (12, 13, 14, 15) ?
bye the way joann and i have moved to richmond so that we dont have to dirive so far every day, i am staying with steve crocker weekdays in d.c. and going home for weekends. the new address is
1716 Wake Forest Drive
Richmond, Virginia 23226
phone: (804) 285-9400
hope things are well with you & our friends in los angeles (and Santa Monica!)

1

(J30234) 15-MAR-74 18:12; Title: Author(s): Jonathan B. Postel/JBP;
Distribution: /LYNN; Sub-Collections: NIC; Clerk: JBP;

ADD 16-MAR-74 19:27 30235

comments on NDSP

Dave: 1

I approve of and support your NSDP proposal. Perhaps the following thoughts will contribute to reducing the ugliness problem: 2

There are certain environments which are controlled enough to not require an escape to the standard, ie CCL, and potentially FTP and RJE. In this case, though, you would want an escape to the local syntax, perhaps a keyword (LOCAL, PATHNAME, ??) or <L-delim> <text> <R-delim> with no keyword. 2a

Would it be possible to modify the syntax to allow something like 2b

DIR[>udd>CNet>anonymous>Owen]FILE[sample_name] instead of: 2b1

DIR[]DIR[udd]DIR[CNet]DIR[anonymous]DIR[Owen]FILE[sample_name] ? 2b2

In fact, the former seems to be allowed by your syntax, but not supplied with an interpretation. I can't think of any problems with this. 2c

Buz 3

ADO 16-MAR-74 19:27 30235

comments on NDSP

(J30235) 16-MAR-74 19:27; Title: Author(s): A. D. (Buz) Owen/ADO; Distribution: /DHC JBP MAP(fyi); Keywords: NDSP, pathnames, CCL, FTP, RJE; Sub-Collections: NIC; Clerk: ADO;

ADO 16-MAR-74 20:06 30236

on protocols and the RSEXEC

Jon: 1

The RSEXEC supports some operations which can't be accomplished reasonably with FTP, but which could be with some limited extensions. These are copy and append where both files involved are files local to the server. One possible implimentation might follow the model of rename. 2

RSEXEC also supports linking and a WHO function, which have no place in FTP or RJE, but which would fit easily into an executive protocol. 3

I think that at least the former idea is protocol worthy. Any comments? 4

Buz 5

ADO 16-MAR-74 20:06 30236

on protocols and the RSEXEC

(J30236) 16-MAR-74 20:06; Title: Author(s): A. D. (Buz) Owen/ADO; Distribution: /JBP RHT; Keywords: FTP copy, FTP local append, WHO function, linking; Sub-Collections: NIC; Clerk: ADO;

SAFE AND SOUND AT DISNEYLAND 1

SAFE AND SOUND AT DISNEYLAND

(J30242) 18-MAR-74 16:10; Title: Author(s): Carl A. Sunshine/CAS;
Distribution: /VGC; Sub-Collections: NIC; Clerk: CAS;

VGC 18-MAR-74 16:37 30243

Partial message to Carl(I typed <CR> too soon)

Carl, type NO RAISE to TENEX monitor to get it to stop echoing uppercase characacters for lowercase ones. It is more sensible for the operating system to halndle this task, as Yogen suggested earlier. The TECO file created when you play the link game will be deleted when you log out since it is a temporary file. You would have to specially change its name to prevent it from being deleted upon logging out. you can ask for a print out of the temporary file by typing "TYPE <filename> <CR>" 1

VGC 18-MAR-74 16:37 30243

Partial message to Carl(I typed <CR> too soon)

(J30243) 18-MAR-74 16:37; Title: Author(s): Vinton G. Cerf/VGC;
Distribution: /CAS VGC; Sub-Collections: NIC; Clerk: VGC;

lynn:
could you please track down and find out and let me know whats happpening about publishing my thesis as a technical report? there are many people who i have told that it will be coming out soon so i would like to be assured as to the truth of my promisses.
please add the following name to the list of those it should be distributed to: K. Elanadhan / Department of computer Science / State University of New York / Stonybrook, NY. 11790.
thanks,
--jon.

1

(J30244) 19-MAR-74 08:37; Title: Author(s): Jonathan B. Postel/JBP;
Distribution: /LYNN; Sub-Collections: NIC; Clerk: JBP;

buz:
thanks for your messages the other day on executive protocols and the network standard pathname proposal, Bob thomas was curious as to your interest in the rsexec protocol, which by the way is documented in a file at bbn in BTHOMAS, let me try a nsdp name for it
%Host/BBN/DIR<BTHOMAS>FILE/IECPTCL/TYPE/DOC/
or in tenex notation
<BTHOMAS>IECPTCL.DOC

In any case thanks for your comments, i may not be able to act on them very fast but they will be considered when i do. i am trying to set aside some tiime to think out the issues involved in executive protocols and the proper place for ftp, rje, graphics in such an environment,
--jon.

(J30245) 19-MAR-74 08:46; Title: Author(s): Jonathan B. Postel/JBP;
Distribution: /ADD; Sub-Collections: NIC; Clerk: JBP;

mike:
where is the host file as advertized in rfc's and such? i just read the blurb in <nic>hostnames.txt but couldnt even login as it directs there, what goes?
--jon.

1

(J30246) 19-MAR-74 09:05; Title: Author(s): Jonathan B. Postel/JBP;
Distribution: /MDK; Sub-Collections: NIC; Clerk: JBP;

Bill:
please ask that i be added to the network measurement group.
thanks
--jon.

1

(J30247) 19-MAR-74 09:29; Title: Author(s): Jonathan B. Postel/JBP;
Distribution: /WEN ALC LYNN MLK; Sub-Collections: NIC; Clerk: JBP;

fy-74 contract dollars requirements

the following efforts have been funde from project 5550:

pr number	title	fy-74	fy-75
b-4-3225	semanol j73	30,000	69,000
b-4-3233	compiler opt study	10,000	32,000
b-4-3229	soft mod study	30,000	-0-
b-4-3232	struct prog sys	80,000	252,713
b-4-3269	impl jocit j3 comp	35,000	-0-

the following efforts have been funded in fy-74 from project 5581;

pr number	title	fy-74	fy-75
b-4-3230	ext of harvard ecl	20,000	50,000
b-4-3245	gcos invest (tpap)	10,000	80,000
b-4-3247	gcos simscript mod	10,000	85,000
b-4-3269	impl jocit j3 comp	35,000	-0-

john giordano has asked frank hadynski for $35,000 more to fund the additional 2 efforts in project 5581;jocit maintenance and gcos simscript. if he oks this we will have funded all our efforts in project 5581 that we planned in fy-74.

in paragraphs 2, and 9 the zero funding in fy-75 means that we will fund these efforts from project 5581 in fy-75.

PR Statistics 19 Mar 74

(J30248) 19-MAR-74 10:00; Title: Author(s): Thomas J. Barkalow/TJB;
Distribution: /RFI RAL JLM FJT ACD; Sub-Collections: NIC; Clerk: EJK;

JCN 19-MAR-74 11:14 30249

Note re Journal Delivery to Perry Directory at Office-1

John: I need to be quite sure that you are getting Journal delivery and I'd also like some idea of the time it takes during a busy day such as this is. Let me know (by sndmsg for speed?) when and if you read this? and the time it seems to have appeared in your directory? Thanks Jim Norton 1

JCN 19-MAR-74 11:14 30249
Note re Journal Delivery to Perry Directory at Office-1

(J30249) 19-MAR-74 11:14; Title: Author(s): James C. Norton/JCN;
Distribution: /JSP CKM JDH; Sub-Collections: SRI-ARC; Clerk: JCN;

DAY 19-MAR-74 13:03 30250

Change of Address

To All, 1

This note is to inform all of you of the change of my Net mailing address. From now on, Net mail for me should be sent to Day (or DAY) at MIT-Multics. 2

John Day 3

Change of Address

(J30250) 19-MAR-74 13:03; Title: Author(s): John D. Day/DAY;
Distribution: /USING FTPIG; Sub-Collections: NIC USING FTPIG; Clerk:
DAY;
Origin: <ILLINOIS>NOTE.NLS;1, 19-MAR-74 12:45 DAY ;

DAY 19-MAR-74 13:13 30251

Change of address

Marcia, 1

I would like to inform you of a change in my network mailing address for both the ARPANET directory and the distribution of journal citations, etc. Mail for me should now be sent to Day (or DAY) at MIT-Multics. 2

Also i would like to point out a minor error in the directory. Under the list of programs, PEESPOL is listed as an assembler. This is not the case. Peespol is a compiler and a language. It may sound like nit-picking (and it is), but we would like it changed. 3

Thanks much,
John Day 4

For your information, suggest you use the command Print Plex .1 with View specs xm for an outline.

STANDARD FORMATS 1

1a

Reply To
Attn of: CS/3008 20 Feb 1974 1b

Subject: Standard Formats 1c

To: IS 1d

Since arriving at RADC, Colonel L. Giesy has asked on several occasions that material be submitted to him in a standard format such as a talking paper. There was little precedence for this at RADC and very little information on the proper format. Standard Air Force Systems Command formats have been obtained from the Director of Science and Technology. The instructions for the use of these formats are incorporated on the form and have been modified slightly to meet RADC requirements. 1e

1f

1g

FRANCIS R. O'CLAIR, Colonel, USAF 1 Atch 1h

Director of Support Services Standard Formats 1i

POINT PAPER 2

2a

(Put Title of Subject on This Line) 2b

A Point Paper provides a summary of a program/project/problem in outline form, FOR THE COMMANDER. The Paper must include key issues. (The Writer may employ considerable candor in preparing a Point Paper since distribution is limited to the Command Section). 2c

- Point Papers are in OUTLINE FORM, NOT narrative 2c1
- Should be clear, concise, complete, timely 2c1a
- Complete sentences are not required 2c1b
- Can be in first person 2c1c
- Should not include attachments 2c1d

- Organize body of paper as follows: 2c2
  - 1. Status: 2c2a
  - 2. Issues: 2c2b
  - 3. Positions: (Include Reasons for Talking Position, if known) 2c2c
    a. RADC: 2c2c1
    b. Others: (Specify Who) 2c2c2

(NOTE: Indicate Positions AFTER Each Issue). 2d

- Margins will be 2d1
  - One and one-half inches at top, bottom, and left side 2d1a
  - one inch at right side 2d1b
- Paper - plain bond, 8 x 10 1/2 inches 2d2
- Indicate date prepared in lower left corner (as below) 2d3
- RADC Staff Offices should use 2 or 3 Letter Symbol in the lower right-hand corner. 2d4

2d5

2d6

As of: Prepared by: 2e

BACKGROUND PAPER 3

3a

(Put title of subject on this line in lower case type) 3b

1. The purpose of a background paper is to provide the Commander, or other key personnel, with background information on a designated subject. The paper should include essential facts, current RADC and/or Air Force positions, important developments, observations, cautions, recommendations, etc. 3c

2. RADC background papers can be requested on any subject and are frequently used to amplify or explain the contents of designated documents such as briefing synopses, letters, information

summaries, news articles, congressional papers,etc. When used to amplify or explain another document, the background paper should not duplicate the data contained in the target document. 3d

3. Background papers are not intended to be placed in conference brochures. They are prepared to provide specific information for the Commander, or other designated person, and are not otherwise distributed. A background paper conveys information only and is not intended to be used as a talking paper. 3e

4. The background paper should be clear, concise, complete and timely. It should be written in the third person and confined to one page if possible. 3f

5. The title should be underscored as shown above. 3g

6. Margins should be at least one and one-half inches at the top, bottom, and left side and one inch at the right side. 3h

7. Paper should be typed single space on plain bond paper, size 8 x 10 1/2. 3i

8. The symbol of the organization preparing the background paper should be placed in the lower right corner of the page as shown below. RADC staff offices should use a 2 or 3 letter symbol. 3j

3k

3l

As of Prepared by 3m

TALKING PAPER 4

4a

(Put title of subject on this line in lower case type) 4b

The purpose of a talking paper is to provide an outline summary which the Commander will use as a guide in talking extemporaneously. The paper should include background, key points to be covered, pertinent issues, alternatives, recommendations, etc. Following format should be used: 4c

- Talking papers are in outline form, not narrative 4c1

  - Should be clear, concise, complete, timely 4c1a

  - Complete sentences not required 4c1b

- Can be in first person 4c1c

- Title underscored, as above 4c2

- Paper should contain all necessary information 4c2a

- Should normally not include attachments 4c2b

- Margins should be: 4c3

-One and one-half inchess at top, bottom and left side 4c3a

- One inch at right side 4c3b

- Paper double-spaced, on plain bond, size 8 x 10 1/2 4c4

- Indicate acronym or symbol of organization preparing paper: 4c5

- Place in lower right hand corner (as below) 4c5a

- RADC staff offices use two letter symbol 4c5b

4c5c

4c5d

As of Prepared by: 4d

BRIEFING SYNOPSIS 5

5a

(Put title of briefing on this line in case type) 5b

1. The purpose of a briefing synopsis is to describe the essential aspects of a proposed or scheduled oral presentation, 5c

2. Following format should be used to describe the proposed presentation: 5d

BRIEFER - (Briefer's name, rank or grade, telephone number, assigned office, and command) 5d1

LENGTH OF BRIEFING - (Running time, not including discussion) 5d2

PURPOSE OF BRIEFING -(A concise, factual statement of reason for the briefing) 5d3

BACKGROUND - (Brief summary of events leading to briefing) 5d4

SUMMARY OF BRIEFING - (Summarize the presentation in brief outline form as it is to be presented) 5d5

RECOMMENDED ACTION - (Summarize all recommendations which the briefer plans to present for approval) 5d6

3. The briefing synopsis should be written in the third person in a clear concise style and should be confined to one page if possible. 5e

4. The title should be underscored as shown above. 5f

5. Margins should be at least one and one-half inches at the top, bottom, and left side, and one inch at the right side. 5g

6. Paper should be typed single space on plain bond paper, size 8 x 10 1/2. 5h

7. The symbol of the organization preparing the Briefing Synopsis should be placed in the lower right corner of the page as shown below. RADC staff officers should use 2 or 3 letter symbols. 5i

5j

5k

As of Prepared by 5l

INFORMATION SUMMARY 6

6a

(Put title of subject on this line in lower case type) 6b

1. The purpose of an information summary is to provide data which can be placed in a Conference Brochure for information and possible discussion. 6c

2. The body of the Summary should contain all information necessary to present a clear picture of the subject suitable for use by top management. It should provide a complete story so that

it stands alone. Since it provides information only, it should not include recommendations. 6d

3. The information summary should be clear, concise, complete and timely. It should be written in the third person and normally be confined to one page. 6e

4. The title should be underscored as shown above. 6f

5. Margins should be at least one and one-half inches at the top, bottom, and left side, and one inch at the right side. 6g

6. Paper should be typed single spaced on plain bond paper, size 8 x 10 1/2. 6h

7. The symbol of the organization preparing the information summary should be placed in the lower right corner of the page as shown below. RADC staff offices should we a 2 or 3 letter symbol. 6i

6j

6k

As of Prepared by 6l

FORMAT FOR ITEM OF INTEREST 7

7a

UNDERLINE SUBJECT (Classification) 7b

Provide a short narrative which clearly states the problem or event, identifies principals involved and actions taken or pending. 7c

Normally the item submitted is limited to one paragraph. A separate page should be used for each item of interest submitted. 7d

Wording of the statement should recognize the fact that it is addressed to the Commander or Deputy. 7e

No attachments are to be included in your response. 7f

Clearly indicate the security classification of both the title and the document. 7g

7h

7i

Prepared by: 7j

Date: 7k

Classification: 7l

TERMS OF REFERENCE 8

8a

Put title of subject on this line in upper case type 8b

Purpose 8c

The purpose of a terms of reference is to define the need for and requirements of a study, analytical,, or special committee effort. This portion should be a short narrative which clearly states the purpose of the study or analysis to be undertaken. 8c1

Background 8d

Provide a concise narrative presenting the problems and defining the requirements for the effort. Include other background data as necessary to fully describe the past and present situation. 8d1

Scope 8e

Briefly describe the scope of the effort. Discuss the areas to be incorporated and proposed application of the final results. 8e1

Approach 8f

Present a brief outline of the approach to be taken. Describe the method of operation and method to be used in applying the results. 8f1

Organization 8g

Identify the participating organizations/personnel, the structure of the study/analysis team, and any steering groups or committees involved. 8g1

Schedule 8h

List the major milestones of the effort. 8h1

8h2

8i

Note: Terms of Reference need not be limited to one page. 8j

8k

8l

As of Prepared by 8m

Standard Formats For Commander

(J30252) 19-MAR-74 13:22; Title: Author(s): Anna A. Cafarelli/AAC; Distribution: /RADC; Sub-Collections: RADC RADC; Obsoletes Document(s): ; Clerk: AAC;

hi herb this is herb 1

(J30253) 19-MAR-74 13:32; Title: Author(s): Michael A. Padlipsky/MAP; Distribution: /HSH ; Sub-Collections: NIC; Clerk: MAP;

Test message

This is another message test to Carl Sunshine 19 March 1974 at roughly 1330 hours. Let's see how long it takes beore it is inserted into your file -- probably it happens once a day in the evening. 1

CAS 19-MAR-74 13:41 30254

Test message

(J30254) 19-MAR-74 13:41; Title: Author(s): Carl A. Sunshine/CAS;
Distribution: /CAS; Sub-Collections: NIC; Clerk: CAS;